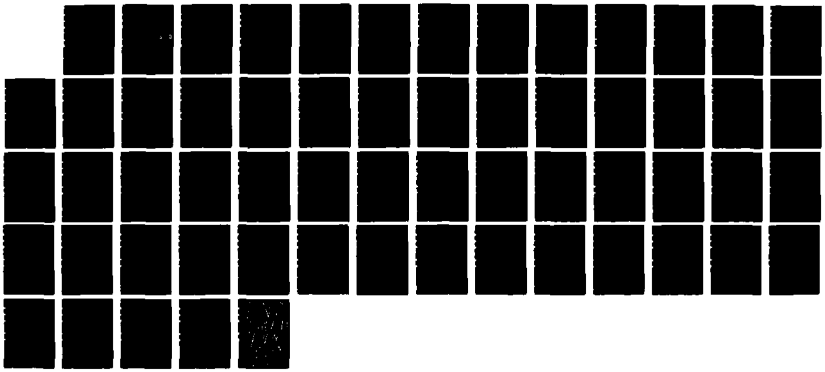
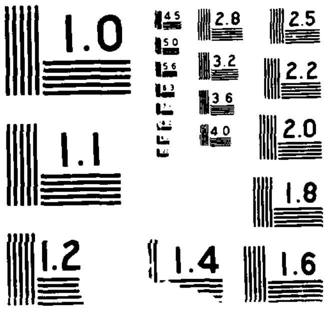



1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.2
1.4
1.6

(4)

FINAL REPORT

FAULT-TOLERANT SIGNAL PROCESSING ARCHITECTURES WITH DISTRIBUTED ERROR CONTROL

Supported by:

THE SDIO INNOVATIVE SCIENCE AND TECHNOLOGY OFFICE
through
GRANT N00014-86-K-0518
OFFICE OF NAVAL RESEARCH
DETACHMENT, PASADENA
1030 EAST GREEN STREET
PASADENA, CA 91106

Principal Investigator:

Professor G. Robert Redinbo
Department of Electrical Engineering
and Computer Science
University of California
Davis, CA 95616
(916) 752-3087

1985

ABSTRACT

Digital filtering architectures that simultaneously offer advantages for VLSI fabrication and contain distributed error-control are presented. Such structures require parallelism as well as inherent error-control capabilities because VLSI implementations are susceptible to temporary and intermittent hardware errors. Three different approaches have been developed to meet these requirements. The first method uses arithmetic decomposition to obtain highly parallel sections each operating with finite field arithmetic while the other two approaches concern finite and infinite convolutions over real or complex number arithmetic domains.

Straightforward realizations depending on highly parallel algebraic decompositions were studied first. They involve the interconnection of fault-tolerant subsystems employing finite field arithmetic into which powerful cyclic error-correcting codes are imbedded naturally. The locations for fault-tolerance and the role of cyclic codes are detailed. Alternative realizations employing finite field transform domains and new techniques for protecting the transform coefficients are developed. These coefficients' special property, called the chord property, permits error detection and correction in the transform domain, and the proper selection of certain code parameters can enhance this capability. Fast transform algorithms with distributed error-control are possible because the interstage variables obey limited chord properties.

Fault-tolerance at the system level in convolution calculations is now possible with recently developed generalized cyclic codes defined over the rings and fields commonly employed for these calculations. Modern high-speed convolvers use sophisticated arithmetic units operating over large finite integer rings or with floating point approximations to the real or complex field. Error control is attached directly to the data permitting protection of any form of parallel or distributed system configuration. New systematic encoding and data manipulation techniques make error detection with generalized cyclic codes straightforward and efficient. The necessary overhead parity computations have complexity proportional to the number of parity symbols squared, whereas the error-detecting capability for both random and burst errors is directly related to this parity number. Numerical error effects are considered with regards to the tolerances necessary in the parity calculation and comparison operation.

Infinite convolutions are protected using the error-detecting capability of real convolutional codes. The normal convolution operation is surrounded with parallel parity channels, and erroneous behavior is detected by externally comparing the calculated and recalculated parity samples thus eliminating any degradation in high-speed operation. A rate (k/n) real convolutional code produces (n-k) parity samples for every k filter samples causing the parity channels to operate at a rate decimated by k. Significant complexity reductions are possible by modifying the code structure, without loss of error protection, yielding simplified parity channels with finite impulse response (FIR) structures operating at rate decimated by k.

A statistical analysis of the modified code's error-detection comparator outputs provides an understanding of its threshold requirements and permits a bound on the comparator's noise variance. The new method involving real convolutional codes is contrasted with another previously proposed method which uses a block code for the states and output in a digital filter's

state-variable realization. A parity comparison is required every sample instant and in general the convolutional code approach is more efficient and flexible.

FINAL REPORT

FAULT-TOLERANT SIGNAL PROCESSING ARCHITECTURES WITH DISTRIBUTED ERROR CONTROL

Supported by:

THE SDIO INNOVATIVE SCIENCE AND TECHNOLOGY OFFICE
through
GRANT N00014-86-K-0518
OFFICE OF NAVAL RESEARCH
DETACHMENT, PASADENA
1030 EAST GREEN STREET
PASADENA, CA 91106

Principal Investigator:

Professor G. Robert Redinbo
Department of Electrical Engineering
and Computer Science
University of California
Davis, CA 95616
(916) 752-3087

INTRODUCTION

As digital electronic implementations of arithmetic units become more dense through shrinking VLSI technology and as their speed of operation increases, fault-tolerance will be needed within arithmetic systems. The VLSI revolution has produced cheap, very high-speed, arithmetic processors which can momentarily introduce errors in a simple calculation, a situation termed soft errors. Error protection against such errors is a critical requirement. Erroneous calculations must be detected as close to their source as soon as possible to avoid propagating incorrect values beyond certain hardware and data boundaries, especially in distributed and parallel computing systems.

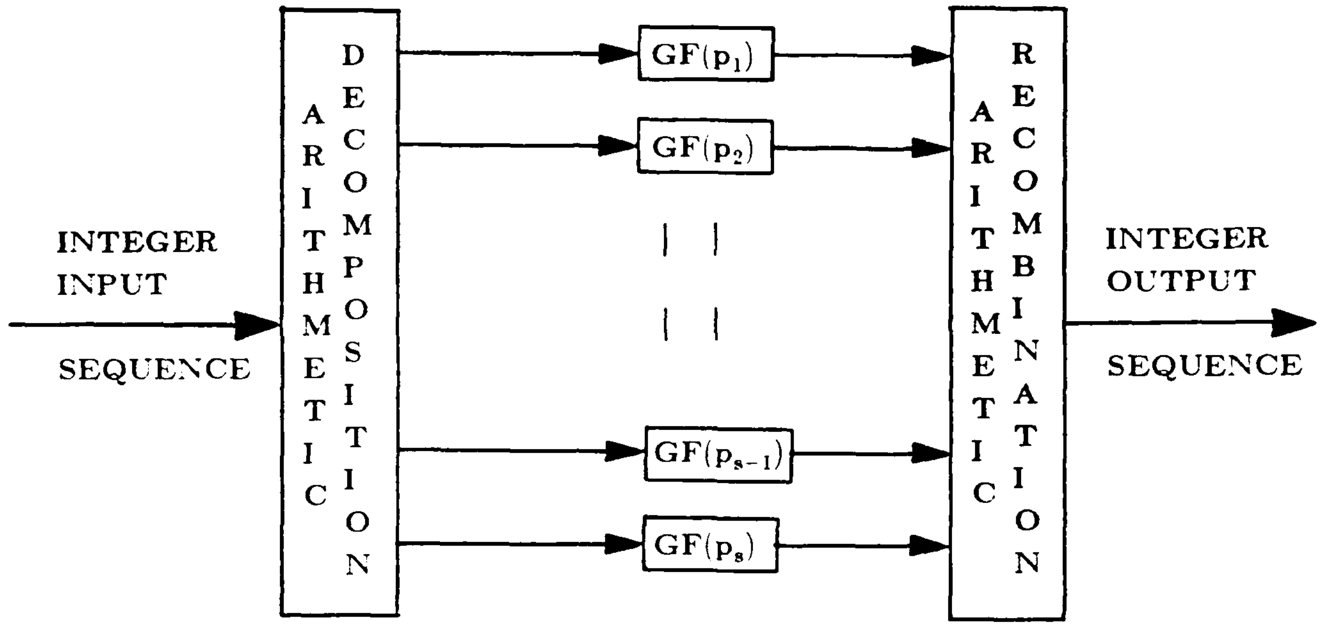
The work and results have covered three different approaches to signal processing system design. The first method employed a maximally parallel decomposition of the underlying arithmetic domain so that many individual processing sections, each operating over a finite field, can process data efficiently in parallel. This allowed classical finite field error-correcting codes to be incorporated independently in each processing path. The other two approaches viewed the arithmetic domain either as a large finite ring or the real or complex field. New, recently developed, error-correcting codes over these larger algebraic structures provide protection for finite or infinite convolutions. Finite length convolutions are protected with real cyclic codes whereas those with infinite length kernels rely upon real convolutional codes.

The results from each of these three approaches will be briefly described in the following sections. The results on the parallel arithmetic decomposition have appeared in the open literature. However, the newer results concerning applications of real codes are still in the publication process. Hence, two very detailed appendices are included to fully document these results.

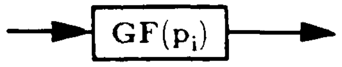
PARALLEL DE COMPOSITIONS

The first work directly embedding data protection within signal processing involved an arithmetic decomposition of the data samples. The Chinese remainder theorem [1] provided the basis for extracting finite field components from numerical samples. This approach distributed cyclic codes, defined over the respective finite fields, throughout the architecture, greatly increasing the overall reliability. These architectures used simple arithmetic and allowed powerful error-control features to be built directly into all the parts of the system because the inherent algebraic structures of error-correcting codes, over general finite fields, match the fundamental operation in filtering convolution. Special structures associated with cyclic codes called minimal ideals have very important algebraic properties, not only affording implicit error protection but leading to fast maximally parallel implementation algorithms.

Signal processing operations, after the proper sampling, scaling, rounding and sequence segmentation are viewed as being performed over a ring of integers modulo M where M is a large positive integer [1]. Furthermore it is possible to decompose the implementation into many parallel realizations by using residue number system techniques. When the modulus M is a product of only distinct primes, the parallel decomposition results in subsections operating with finite field arithmetic requiring very simple arithmetic processing. This situation is depicted in Figure 1,



$M = p_1 p_2 \cdot \cdot \cdot \cdot p_{s-1} p_s \quad : \quad p_i$ PRIME



DENOTES PROCESSING USING
FINITE FIELD ARITHMETIC
BASED ON PRIME p_i

PARALLEL DECOMPOSITION OF PROCESSING
WHEN MODULUS CONTAINS NO MULTIPLE PRIMES
FIGURE 1

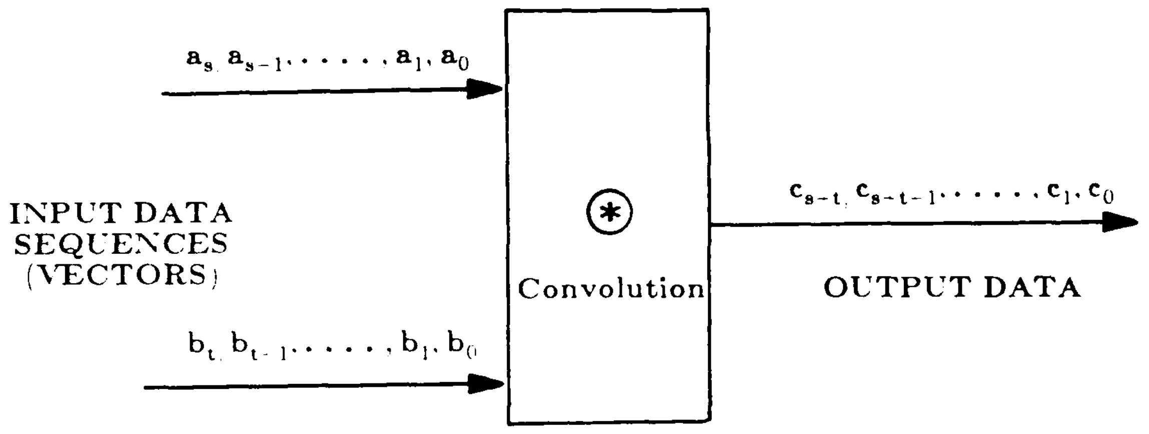
where each parallel subsection operates over the distinct finite fields denoted by GF(p_i), i=1,2,...,s. The arithmetic decomposition and recombination operations shown respectively at the input and output in Figure 1 employ the Chinese remainder theorem. Three papers published in the open literature fully describe this approach [2-4].

The results from this previous research indicated the importance of treating data as significant entities to be protected. While the distributed finite field approach afforded adequate protection, its applicability was restricted and effects of internal errors were difficult to predict because of the arithmetic reconstruction process. However, one outgrowth of this work was the interrelationship between cyclic codes and the important convolution operation.

FINITE CONVOLUTIONS

New published research provided powerful cyclic codes defined over real numbers [5]. In addition, similar codes over finite rings which could be applied to fixed point processing were described in the literature [6]. Concurrently digital electronic technology, principally in the VLSI area, dictated that more arithmetic processing power for a given area was available and floating point processors became widely prevalent.

It became possible to view convolution as an operator on arrays of data, as depicted in Figure 2, and employ cyclic codes over the same arithmetic structure being used in the underlying arithmetic processors. The results of this previous research are detailed in Appendix A and they were reported in two International Conferences [7,8]. (A full-length paper is currently under review for IEEE Transactions on Acoustics Speech and Signal Processing [9].) The basic error-detecting approach computes parity samples affiliated with each array, producing the output parity matching the convolution output. The overhead operations in the parity recalculations are on the order of the number of parity values squared. On the other hand, this number of parity positions is identical with the error-detecting capability of the code. Error protection levels are a relatively small fraction of long data arrays so that the additional complexity is quite low. Hence,



$a_s, a_{s-1}, \ldots, a_1, a_0$
INPUT DATA SEQUENCES (VECTORS)
$b_t, b_{t-1}, \ldots, b_1, b_0$
$\circledast$
Convolution
$c_{s+t}, c_{s+t-1}, \ldots, c_1, c_0$
OUTPUT DATA

$$c_j = \begin{cases} \sum_{i=0}^{s} a_i b_{j-i} \\ \sum_{i=0}^{t} a_{j-i} b_i \end{cases} \quad ; j=0, 1, 2, \ldots, (s+t).$$

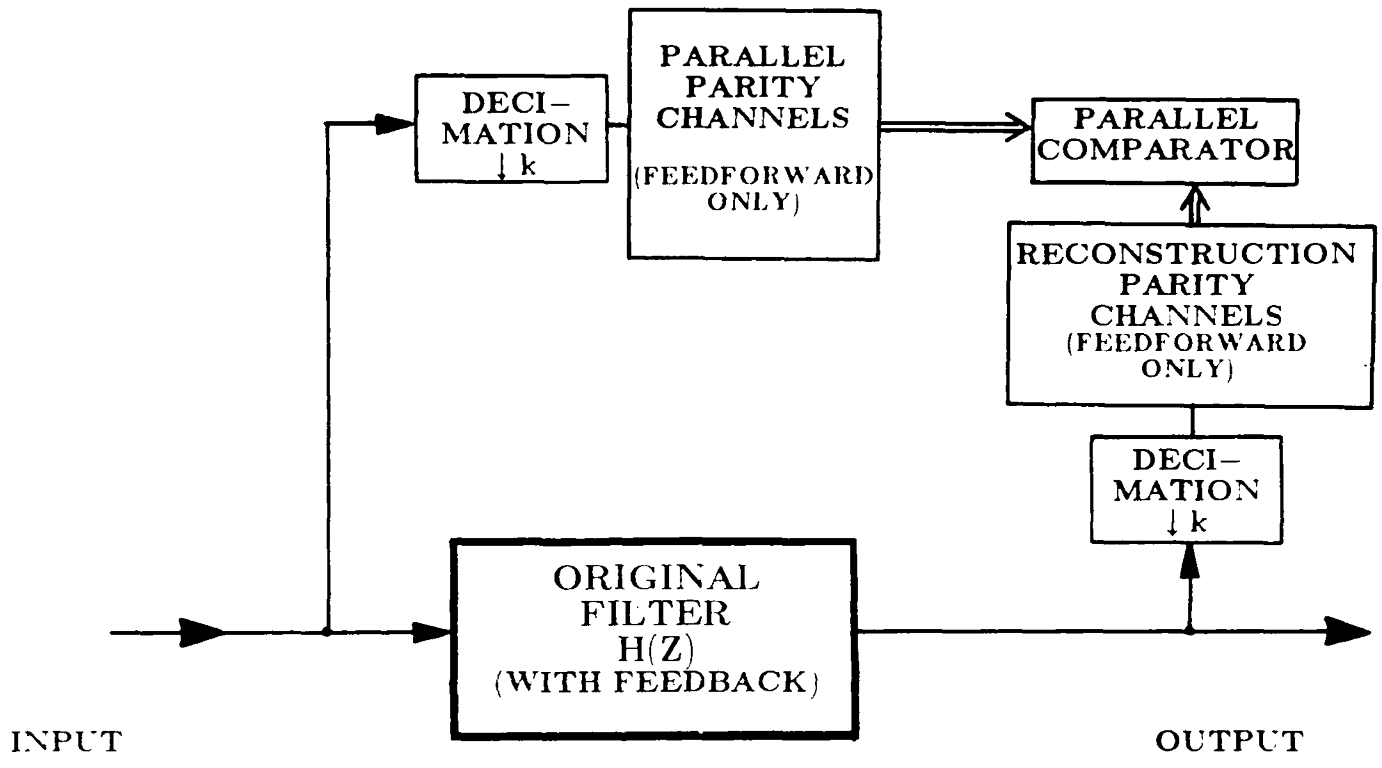
CONVOLUTION OF ARRAYS
FIGURE 2

any filtering or signal processing operations involving finite-impulse-response (FIR) weighting kernels are easily and efficiently protected by this method.

Convolution with finite length kernels are sometimes implemented via fast Fourier transform algorithms; the data are transformed, weighted by the corresponding transform of the kernel, and the resulting component-wise products inverse transformed. However, the overall effect is cyclic convolution wherein the finite length sequences are treated as periodically repeated. Unless padding zero values are inserted at one end of the sequences there will be contributions from overlapping segments. This is called cyclic convolution. Nevertheless this only slightly complicates the error protection method using cyclic codes. The end-around effects inherent in cyclic convolution must be separated and properly handled in the parity channel calculations. The additional complexity is related to number of parity positions times the length of the shortest sequence involved in the convolution. (See Appendix A.)

INFINITE CONVOLUTIONS

Another widely used form of signal processing employs input and weighting sequences that are essentially infinite. They are usually realized with feedback paths and temporary storage registers, and therefore present a different error protection challenge. Convolutional codes are well-matched to this form of signal processing. But in a straightforward implementation using parallel parity channels which compute the corresponding parity values the hardware and storage complexity are comparable to the original realization. Even though the computational rates in the parity channels are slower, the net increase in overhead is unacceptable. However, real convolutional codes have extra degrees of freedom which permit significant simplifications in the parity channels without reducing the error-detecting capabilities. (See Appendix B.) The parity computations and recalculations use FIR realizations operating at reduced rates, lowering the error-protecting overhead. This approach is outlined in Figure 3 where both parity channel groups have effectively decimated input streams [10]. The theoretical basis for this method of protecting infinite weighting signal processing is detailed in Appendix B. Preliminary results were presented at an



EXTERNALLY PROTECTED FILTER SYSTEM
FIGURE 3

APPENDIX A

PROTECTING CONVOLUTION-TYPE ARITHMETIC ARRAY CALCULATIONS WITH GENERALIZED CYCLIC CODES

INTRODUCTION

As digital electronic implementations of arithmetic units become more dense through shrinking VLSI technology and as their speed of operation increases, fault-tolerance will be needed within arithmetic systems. The VLSI revoiution has produced cheap, very high-speed, arithmetic processors which can momentarily introduce errors in a simple calculation, a situation termed soft errors. Error protection against such errors is a critical requirement. Erroneous calculations must be detected as close to their source as soon as possible to avoid propagating their effects beyond certain hardware and data boundaries, especially in distributed and parallel computing systems.

High-speed convolution of data arrays is one common and important class of arithmetic processing which needs adequate protection. Properly defined linear codes can protect the addition and scaling of data arrays. However the convolution of such arrays while constructed from there simple operation require more structured codes. This paper demonstrates how generalized cyclic codes, defined over the rings and fields usually employed in such processing, may be incorporated directly and quite naturally within the implementations of such arithmetic systems. The integrity of the output data is of paramount importance whether the fundamental operations are shared among distributed or parallel processors, or concentrated in a single special unit. Furthermore, new encoding and parity manipulation methods are developed which permit straightforward and efficient mechanizations. Errors are detected immediately at the conclusion of the processing pass, allowing appropriate error control actions to be initiated. Typical responses to detected errors may be to retry the calculation, reconfigure the overall system, or enter a subsystem testing mode.

The central operation in signal processing or digital filtering is the convolution of data sample sequences. These samples' arithmetic values may be viewed as algebraic elements in finite rings such as the integers modulo an integer q, denoted by Z_q, or as real or complex numbers, labeled respectively by

R and C. Typical ring representations include two's complement, where $q = 2^m$ or one's complement with $q = 2^m - 1$. The machine format can be either fixed or floating point.

The fundamental convolution operation may be displayed using mathematical symbols, starting from the two data sequences.

$$\text{DATA SEQUENCES} \begin{cases} a_0, a_1, a_2, \ldots, a_s \\ b_0, b_1, b_2, \ldots, b_t \end{cases} ;\ a_i, b_j, \varepsilon\, Z_q \text{ or R or C}$$

Their convolution involves the well-defined operations in the underlying ring or field.

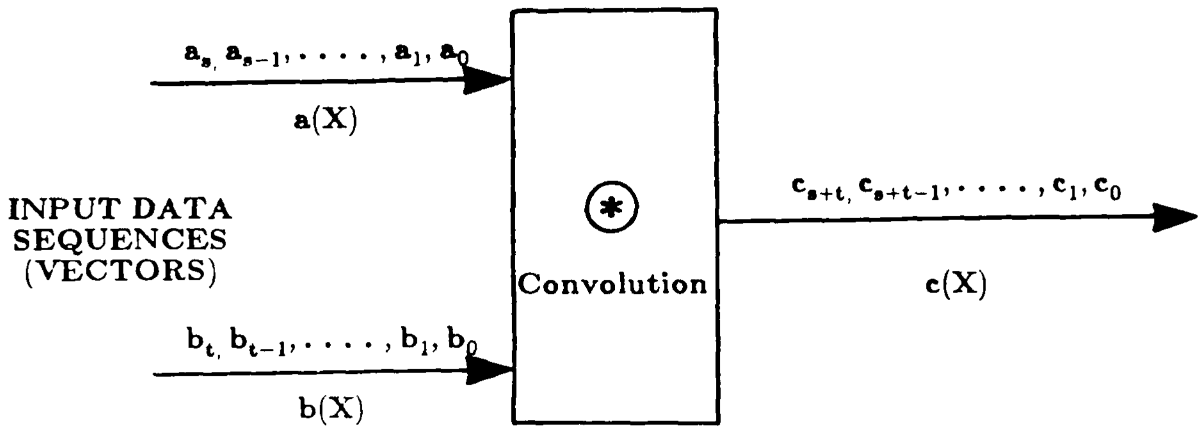
$$\begin{aligned} c_j &= \sum_{i=0}^{s} a_i b_{j-i} \\ & \qquad\qquad j = 0,1,2, \ldots, (s+t). \\ c_j &= \sum_{i=0}^{t} a_{j-i} b_i \end{aligned} \tag{1}$$

In these defining equations, any sample with index outside the prescribed range is considered to be zero. See Figure 1 for a schematic representation of the basic operation to be protected.

The convolution of a kernel function with a semi-infinite input data stream may be accomplished by several methods. Two popular ones, overlap-add and overlap-save [1,2], segment the input data stream into sections and perform the required convolution with a finite length kernel, preserving pieces of the resulting outputs to be recombined into a single continuous output stream. Nevertheless the fundamental approach remains the convolution of finite sections as described in equation (1).

A modern view and the main point of departure for several recent texts [1-3] considers these sequences as polynomials in an indeterminant X. Then convolution is intrinsic in the normal definition of polynomial products.

$$\begin{aligned} & c(X) = a(X)b(X) \\ & \text{where} \\ & a(X) = a_0 + a_1 X + a_2 X^2 + \ldots + a_s X^s \leftrightarrow \{a_0, a_1, \ldots, a_s\} \\ & b(X) = b_0 + b_1 X + b_2 X^2 + \ldots + b_t X^t \leftrightarrow \{b_0, b_1, \ldots, b_t\} \\ & c(X) = c_0 + c_1 X + c_2 X^2 + \ldots + c_{s+t} X^{s+t} \leftrightarrow \{c_0, c_1, \ldots, c_{s+t}\} \end{aligned} \tag{2}$$


$a_s, a_{s-1}, \ldots, a_1, a_0$
$a(X)$
INPUT DATA SEQUENCES (VECTORS)
$\circledast$
Convolution
$c_{s+t}, c_{s+t-1}, \ldots, c_1, c_0$
$c(X)$
$b_t, b_{t-1}, \ldots, b_1, b_0$
$b(X)$

$$c(X) \longleftrightarrow c_j = \begin{cases} \sum_{i=0}^{s} a_i b_{j-i} \\ \sum_{i=0}^{t} a_{j-i} b_i \end{cases} \quad ; j=0, 1, 2, \ldots, (s+t).$$

CONVOLUTION OF ARRAYS
FIGURE 1

The polynomials in these cases belong to an algebraic structure, the commutative ring of polynomials, usually given the respective symbols $Z_q[X]$, $R[X]$ and $C[X]$.

Many fast signal processing algorithms rely upon the mathematical properties arising from this polynomial view. For example, when an exponent k is chosen sufficiently large, an equivalent form of equation (2) makes the algebraic structure even richer by introducing residue class rings modulo (X^k-1), [1].

$$c(X) \equiv a(X)b(X) \text{ modulo } (X^k-1)\ ;\ k \geq s+t+1 \tag{3}$$

The potential for protecting such operations with cyclic error-correcting codes is obvious in light of this equation. Cyclic codes are defined and manipulated as polynomial residue algebras, and their common fundamental processing operation involves polynomial products [4]. Thus it is natural to investigate cyclic codes as a powerful means of detecting errors in these types of operations.

In order to apply cyclic codes to the arithmetic setting being considered here, two hurdles must be overcome. Firstly, most cyclic codes are defined over finite fields, primarily because their design depends upon roots of polynomials in extension fields. Secondly, and equally as important, no previously known data encoding format exists which leaves the data symbols in their unaltered form while appending the proper parity symbols when passed through the polynomial product operation. The data and parity positions become intermixed when processed this way. The first difficulty is resolved by generalized cyclic codes which have been studied only recently [5,6]. These types of codes will be motivated and detailed below. In addition, real cyclic codes for use with floating point formats will be explained including examples. The second problem is solved by a new systematic encoding approach.

GENERALIZED CYCLIC CODES

Cyclic codes represent a powerful and wide class of codes with easily determined guaranteed distance properties that can be used for detecting both random and burst errors. They are naturally defined in a residue class ring of a polynomial algebra using the modulus (X^n-1), where n is the code length [4]. One feature guarantees that every cyclic end-around shift of the elements comprising a code polynomial is also a code polynomial, the significance of the modulo (X^n-1) reduction.

Generalized cyclic codes will be explained using a generic algebraic structure Γ which is at least a commutative ring with identity. Such a structure also covers the fields of real numbers R and complex numbers C. The ring of polynomials $\Gamma[X]$, the set of polynomials in indeterminant X, can be reduced to a residue class residue class ring using (X^n-1), [4]. This new structure is written symbolically as $\Gamma[X]/(X^n-1)$.

A generalized cyclic code is defined by a single generator polynomial, g(X), whose leading term is a unit in Γ. The code is a principal ideal generated by g(X), denoted by ((g(X))), and formally defined as

$$((g(X))) = \{p(X) \equiv q(X)g(X) \text{ modulo } (X^n-1) : q(X)\varepsilon\Gamma[X]\} \tag{4}$$

The degree of g(x) is (n-k), the number of parity positions contained in the code. Construction techniques for the generator polynomial depend on the exact nature of Γ. Nevertheless, the Euclidean Algorithm is a common underlying principle. It guarantees unique quotients and remainders for division, provided that the highest indexed coefficient in the divisor g(X) is a unit (invertible) element of Γ. Since this general result will be cited later, it will be included here [8]; for any $f(X)\varepsilon\Gamma[X]$ there exist polynomials q(X), the quotient, and r(X), the remainder, such that

$$f(X) = q(X)g(X) + r(X) \text{ ; degree } \{r(X)\} < \text{degree } \{g(X)\} \tag{5}$$

The Euclidean Algorithm also shows the burst detecting capabilities of a cyclic code. A burst is a consecutive segment of a code word's elements which has the beginning and ending elements of the segment in error, and permitting any number of erroneous position in between [7]. Since the code is cyclic, a burst can also occur in an end-around sense. A burst can be modeled by adding an error polynomial of the form $X^u e(X)$ to the code word. However a disruption like this can always be detected as long as g(X) does not divide e(X). (Remember every code word by construction (4) is a multiple of g(X), and thus a burst divisible by the generator polynomial is undetectable.) When the polynomial part, e(X), of the burst error polynomial has degree less than that of g(X), this division is impossible, providing the burst detecting ability of (n-k) positions.

Cyclic codes over the real or complex fields are defined by using consecutively indexed powers of the nth complex root of unity, e.g., $[\exp(j2\pi/n)p]$, where $j = \sqrt{-1}$ and p is any integer modulo n. The fundamental construction techniques are given by Marshall [6], and use the discrete Fourier transform domain in which contiguously indexed transform coefficients determine the generator polynomial. By requiring conjugate roots be included, real generator polynomials are constructed. In the more general case of complex numbers, maximum distance separable codes [4] (analagous to powerful Reed-Solomon codes) are easily established. Such codes can detect erroneous positions equal in number to the degree of g(X), the maximum permitted by the Singleton bound [4].

Generalized cyclic codes over finite integer rings, Z_{p^m}, can be defined by first examining roots in some extension field of the finite field Z_p, p a prime number [5]. Primitive elements in this extension field are studied as members of a multiplicative cyclic group in Z_{p^m}. Again, consecutively indexed roots and their conjugates are used in constructing the generator polynomial with the desired error-detecting parameters. The most general situation for Z_q involves fields and rings associated with the prime factors in the integer q. The Chinese Remainder Theorem [1] allows components over Z_{p^m} to be reassembled, defining a generator polynomial over Z_q. The lengths of segments of adjacently indexed roots guarantee the detecting performance of the final code.

Real cyclic codes, primarily applicable to floating point arithmetic formats will be examined in slightly more detail so as to better exemplify the parity operations to be discussed later. They are constructed using consecutively indexed primitive roots of unity [6] and were original called Discrete Fourier Transform (DFT) codes [14,15]. Powers of the nth complex root of unity, W, define the roots of the code generator polynomial g(X).

$$g(X) = \prod_{r \in R} \left(X - W^r \right) : W^{\frac{j2\pi}{n}}$$

R = INDEX SET OF CONSECUTIVE INTEGERS MOD n

$$R \subset \{0,1,2,\ldots\ldots,(n-1)\}$$

The span (number of consecutive indices) determines the error-detecting capability of the code and is the maximum allowable for a linear code [4]. In this regard they closely resemble the BCH and Reed-Solomon codes defined over finite fields [4,7]. The code can detect up to $|R|$ symbol positions in error considering the roundoff tolerance of the comparison operation.

However if the index root set is symmetric about 0 (and including 0) or about $n/2$ (and including $n/2$ if n is even), conjugate root pairs appear in g(X), giving it real coefficients. This restriction narrows the range of parameters permitted in the code slightly (see Property 3 of [6]), but does not reduce the error protection levels in any way. In addition, the real coefficients of g(X) are also symmetric or anti-symmetric about the degree midpoint halving the number of multiplicative operations needed in parity calculation and re-calculation.

A power of the primitive complex root, W^m, where m is relatively prime to n can be used in place of W in the definition of g(X). A different code results with the same error-detecting capability but with roots located at more widely dispersed points on the unit circle. This eases the accuracy requirement in calculating and using the coefficients in g(X). A simple example for a code with length $n = 1024$ and information capacity $k = 1003$ using an index scaling factor $m = 47$, has capability of detecting up to 21 positions in error or any burst up to length 21. Figure 2a displays the generator polynomial while in contrast Figure 2b shows the generator polynomial for the same quality code but with root index scaling factor $m = 23$. The coefficients in the latter example are larger and have a wider magnitude range. Equally good codes may be centered about $-1 = W^{512}$, but they have different coefficient signs since (X+1) is the only linear factor.

These techniques along with others [9-12] insure the availability of a variety of generalized cyclic codes that can provide a wide choice of random and burst error-detecting abilities. With the existence of good cyclic codes over the proper rings and fields established, the second problem of efficient systematic encoding and parity manipulation of the code word symbols will be addressed.

SYSTEMATIC DATA ENCODING AND PARITY MANIPULATION

One important requirement of any protection scheme is the location and manipulation of the original data and associated parity symbols without additional processing. There are several methods for encoding data code words with this feature, generally referred to as systematic encoding [7]. However when two such encoded code words are producted (to implement the convolution), the corresponding parity symbols are not easily distinguished except with complicated processing. This section will first demonstrate a standard systematic encoding technique from which the respective data and parity may be extracted easily. Then a new approach will be given for processing and combining the respective parity parts to yield the new parity values corresponding to the convolved data segments.

A common systematic encoding scheme relies on the Euclidean Algorithm for uniquely defining the parity positions [7]. The data portion, say a(X), is placed in the higher indexed positions, by multiplying by X^{n-k}, effectively shifting to data to inclusively indexed positions, (n-k), (n-k+1), . . . , (n-1). The uniquely

$$
\begin{aligned}
g(X) = {} & -1.000000e{+}00 \quad +7.882733e{-}01\,X^{1} -7.063940e{-}01\,X^{2} \\
& +6.591348e{-}01\,X^{3} -6.277460e{-}01\,X^{4} +6.055386e{-}01\,X^{5} \\
& -5.894287e{-}01\,X^{6} +5.777794e{-}01\,X^{7} -5.696514e{-}01\,X^{8} \\
& +5.644898e{-}01\,X^{9} -5.619806e{-}01\,X^{10} +5.619806e{-}01\,X^{11} \\
& -5.644898e{-}01\,X^{12} +5.696514e{-}01\,X^{13} -5.777794e{-}01\,X^{14} \\
& +5.894287e{-}01\,X^{15} -6.055386e{-}01\,X^{16} +6.277460e{-}01\,X^{17} \\
& -6.591348e{-}01\,X^{18} +7.063940e{-}01\,X^{19} -7.882733e{-}01\,X^{20} \\
& +1.000000e{+}00\,X^{21}
\end{aligned}
$$

Generator Polynomial With Root
Index Scaling m=47.
Figure 2a.

$$
\begin{aligned}
g(X) = {} & -1.000000e{+}00 \quad +1.412738e{+}01\,X^{1} -9.916219e{+}01\,X^{2} \\
& +4.595110e{+}02\,X^{3} -1.575823e{+}03\,X^{4} +4.249810e{+}03\,X^{5} \\
& -9.350972e{+}03\,X^{6} +1.719101e{+}04\,X^{7} -2.682654e{+}04\,X^{8} \\
& +3.590434e{+}04\,X^{9} -4.147518e{+}04\,X^{10} +4.147518e{+}04\,X^{11} \\
& -3.590434e{+}04\,X^{12} +2.682654e{+}04\,X^{13} -1.719101e{+}04\,X^{14} \\
& +9.350972e{+}03\,X^{15} -4.249810e{+}03\,X^{16} +1.575823e{+}03\,X^{17} \\
& -4.595110e{+}02\,X^{18} +9.916219e{+}01\,X^{19} -1.412738e{+}01\,X^{20} \\
& +1.000000e{+}00\,X^{21}
\end{aligned}
$$

Generator Polynomial With Root
Index Scaling m=23.
Figure 2b.

$n = 1024 \quad ; \quad k = 1003$.

REAL CYCLIC CODE GENERATOR POLYNOMIAL EXAMPLES
FIGURE 2

related parity symbols are represented by the polynomial $r_a(X)$, derived from equation (5) with g(X) as the divisor.

$$\{X^{n-k}a(X)\} = q_a(X)g(X) + r_a(X)\ ; \qquad \deg r_a(X) < \deg g(X) \qquad (6)$$

The code word affiliated with data a(X) is given by

$$a(X) \xrightarrow{\text{ENCODE}} [X^{n-k}a(X) - r_a(X)]$$

A simple transposition in equation (6) shows that this is indeed a multiple of g(X), the defining property of a cyclic code word (see equation (4)). Furthermore the parity values represented by $r_a(X)$ do not interfere with the original data, now shown as $X^{n-k}a(X)$, in their shifted positions. A similar encoding also applies to data b(X), where $r_b(X)$ is the unique remainder analogous to equation (6).

$$b(X) \xrightarrow{\text{ENCODE}} [X^{n-k}b(X) - r_b(X)]$$

The protection of the convolution of two code words is considered. If the two respective code words for a(X) and b(X) are producted, it is easy to see the intermingling and overlapping of parity and data parts. However the data portions are easily extracted and producted. Then the question is: how can the parity parts $r_a(X)$ and $r_b(X)$ be processed to yield the correct parity? In symbols, what relationship exists between $r_a(X)$ and $r_b(X)$, and the new parity $r_{ab}(X)$ related to the product a(X)b(X)?

$$[a(X)b(X)] \xrightarrow{\text{ENCODE}} \{X^{n-k}[a(X)b(X)] - r_{ab}(X)\}$$

This new parity part is the remainder in the division by g(X).

$$r_{ab}(X) \equiv X^{n-k}[a(X)b(X)] \bmod g(X) \qquad (7)$$

The answer to be demonstrated in the next paragraph is computationally straightforward.

$$r_{ab}(X) \equiv f(X)r_a(X)r_b(X) \bmod g(X) \qquad (8a)$$

where

$$f(X) \equiv X^k \bmod g(X)\ . \qquad (8b)$$

The validity of the above claim revolves around showing that the expression $\{X^{n-k}[a(X)b(X)]-r_{ab}(X)\}$ is a multiple of g(X), modulo (X^n-1). In this regard two identities need to be compiled. The first comes from equations (8) which imply that there is a quotient $q_{ab}(X)$ satisfying

$$r_{ab}(X) = X^k r_a(X) r_b(X) - q_{ab}(X) g(X) \tag{9}$$

The second needed expression follows from the coded form of b(X), similar to equation (6), this time implying another quotient $q_b(X)$.

$$\begin{aligned} b(X) &\equiv x^n b(X) \equiv X^k\{X^{n-k} b(X)\} \bmod (X^n-1) \\ &\equiv X^k q_b(X) g(X) + X^k r_b(X) \bmod (X^n-1) \end{aligned} \tag{10}$$

These identities when combined with equation (6) permit the following series of equalities.

$$\begin{aligned} \{X^{n-k}[a(X)b(X)] - r_{ab}(X)\} &\equiv [q_a(X)g(X) + r_a(X)][X^k q_b(X)g(X) + X^k r_b(X)] \\ &\quad -[X^k r_a(X) r_b(X) - q_{ab}(X) g(X)] \bmod (X^n-1) \\ &\equiv g(X)\{X^k q_a(X) q_b(X) + X^k q_b(X) r_a(X) \\ &\quad + X^k q_a(X) r_b(X) + q_{ab}(X)\} \bmod (X^n-1) \end{aligned} \tag{11}$$

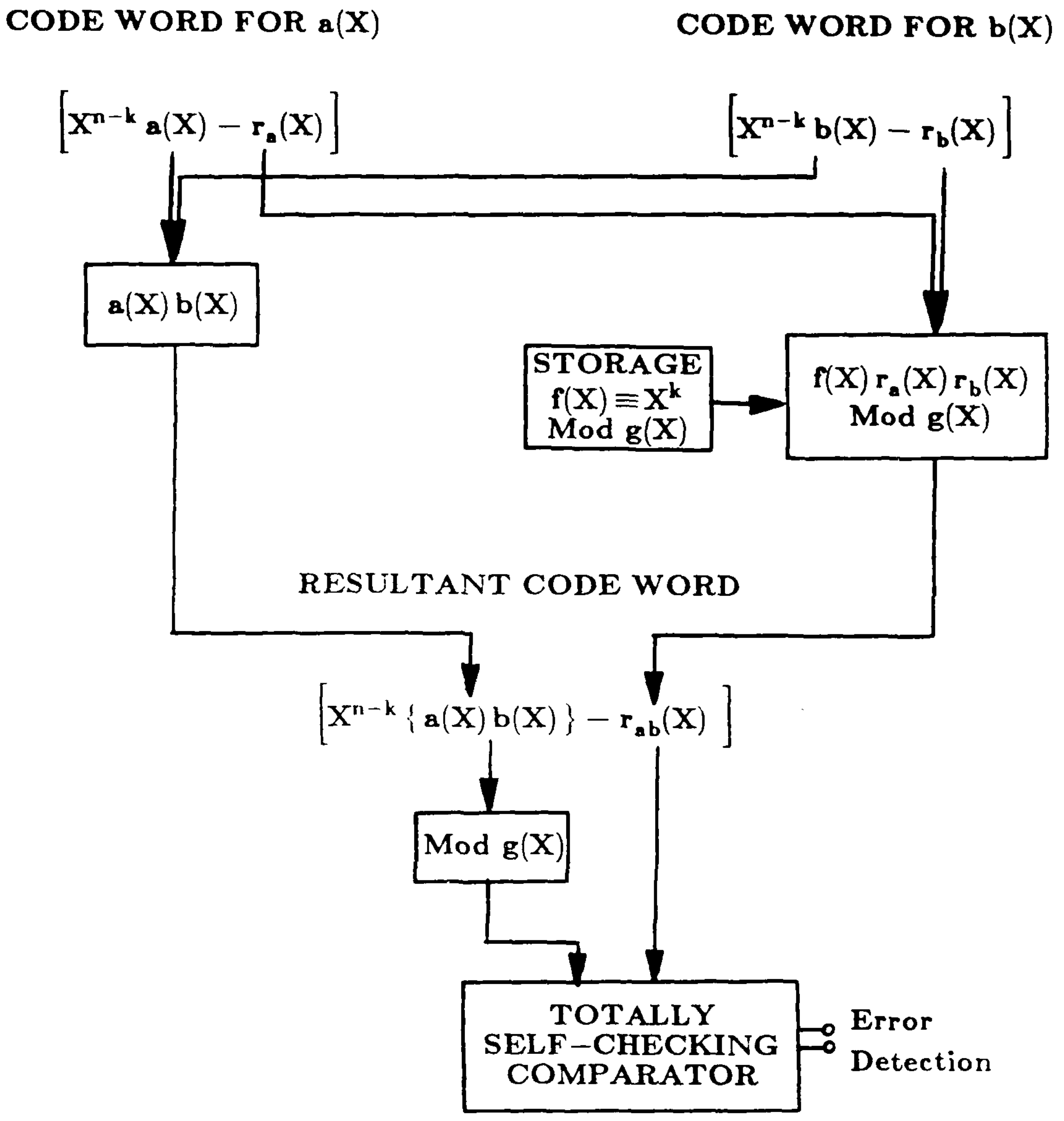
The very construction of $r_{ab}(X)$, equation (8a), guarantees that

$$\deg r_{ab}(X) < \deg g(X) \ .$$

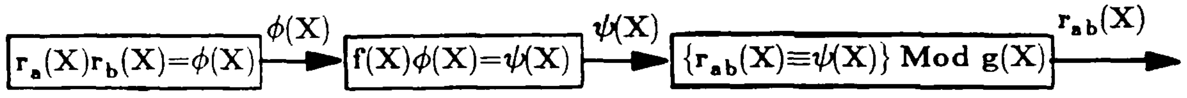
On the other hand, the Euclidean Algorithm asserts a unique remainder polynomial associated with the encoding of [a(X)b(X)]; equations (8) provide that polynomial and it has degree less than (n-k) also.

The use of this systematic encoding in a fault-tolerant realization for convolving sequences represented by a(X) and b(X) is shown schematically in Figure 3. The steps in forming the new systematically encoded code word are easily identified with straightforward manipulations. The protection overhead is governed by the modulo g(X) operations, which in turn are proportional to the degree of g(X), the number of parity positions employed by the code. Even the regeneration of the parity symbols, needed in the totally-self checking comparator [13], depends on the code generating polynomial g(X). The complexity of the modulo reductions will be discussed in the next section.

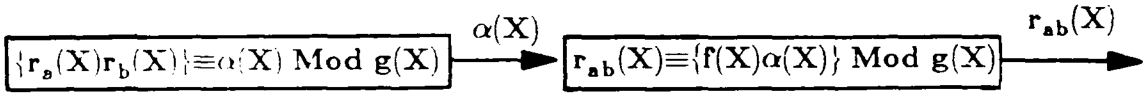
The required parity calculations according to equations (8) may be performed in several orders. The modulo g(X) reduction may be applied after each product or the complete product $f(X)r_a(X)r_b(X)$ may be formed, necessitating more storage, before the modulo reduction is done. Two options are depicted in Figure 4. The parity weighting factor f(X), equation (8b), for the two respective example generator



USE OF SYSTEMATIC ENCODING FORMAT
IN PROTECTING CONVOLUTION $\{a(X)b(X)\}$
FIGURE 3



Products Followed By Modulo Reduction
Figure 4a.


$\{r_a(X)r_b(X)\}\equiv\alpha(X)$ Mod $g(X)$
$\alpha(X)$
$r_{ab}(X)\equiv\{f(X)\alpha(X)\}$ Mod $g(X)$
$r_{ab}(X)$

Intermediate Modulo Reduction
Figure 4b.

ORDER OF PARITY CALCULATIONS
FIGURE 4

$$f(X) \equiv +1.550571e{-01} \quad +1.015542e{+00}\,X^{1} -2.462610e{-01}\,X^{2}$$
$$+1.692161e{-01}\,X^{3} -1.386988e{-01}\,X^{4} +1.218860e{-01}\,X^{5}$$
$$-1.110590e{-01}\,X^{6} +1.034289e{-01}\,X^{7} -9.773097e{-02}\,X^{8}$$
$$+9.330137e{-02}\,X^{9} -8.975209e{-02}\,X^{10} +8.683481e{-02}\,X^{11}$$
$$-8.437486e{-02}\,X^{12} +8.223395e{-02}\,X^{13} -8.028343e{-02}\,X^{14}$$
$$+7.837660e{-02}\,X^{15} -7.630670e{-02}\,X^{16} +7.372199e{-02}\,X^{17}$$
$$-6.991635e{-02}\,X^{18} +6.320080e{-02}\,X^{19} -4.841679e{-02}\,X^{20}$$

X^{1003} MOD g(X)

(g(X) from Figure 2a.)
Figure 5a.

$$f(X) \equiv +2.360182e{+02} \quad 0-3.292485e{+03}\,X^{1} +2.281767e{+04}\,X^{2}$$
$$-1.043690e{+05}\,X^{3} +3.531499e{+05}\,X^{4} -9.391855e{+05}\,X^{5}$$
$$+2.036301e{+06}\,X^{6} -3.685237e{+06}\,X^{7} +5.654087e{+06}\,X^{8}$$
$$-7.428145e{+06}\,X^{9} +8.405427e{+06}\,X^{10} -8.211695e{+06}\,X^{11}$$
$$+6.920363e{+06}\,X^{12} -5.009762e{+06}\,X^{13} +3.090042e{+06}\,X^{14}$$
$$-1.602602e{+06}\,X^{15} +6.846374e{+05}\,X^{16} -2.332175e{+05}\,X^{17}$$
$$+5.998405e{+04}\,X^{18} -1.049345e{+04}\,X^{19} +9.556912e{+02}\,X^{20}$$

X^{1003} MOD g(X)

(g(X) from Figure 2b.)
Figure 5b.

PARITY WEIGHTING FACTOR f(X) EXAMPLES
FIGURE 5

polynomials given in Figure 2 are shown in Figure 5. They are distinguished by the index scaling values of 47 and 23.

There are situations where one of the sequences to be convolved is fixed and known in advance. For example, b(X) could be the impulse response of a digital filter [1,2]. The previous technique can be used to store the known sequence and its precomputed parity positions. For a predetermined sequence represented by b(X), the stored positions correspond to the code word $\{X^{n-k}b(X) - r_b(X)\}$. One simplification allows the parity positions $r_b(X)$ to be combined with f(X), equation (8b), reducing the number of operations required for $r_{ab}(X)$. However, there is an alternate, equally effective method for handling this special circumstance.

The known sequence, say b(X), is also stored in its reduced form modulo g(X). Then the systematically encoded code word related to [a(X)b(X)] is given by

$$X^{n-k}[a(X)b(X)] - r_0(X) \ ,$$

where the parity positions are defined uniquely as

$$r_0(X) \equiv \beta(X)r_a(X) \bmod g(X) \qquad (12a)$$

$$\beta(X) \equiv b(X) \bmod g(X) \qquad (12b)$$

The correctness of this approach is easily demonstrated by noting that equations (12) insure the existence of a quotient $q_0(X)$ and remainder $r_0(X)$ giving

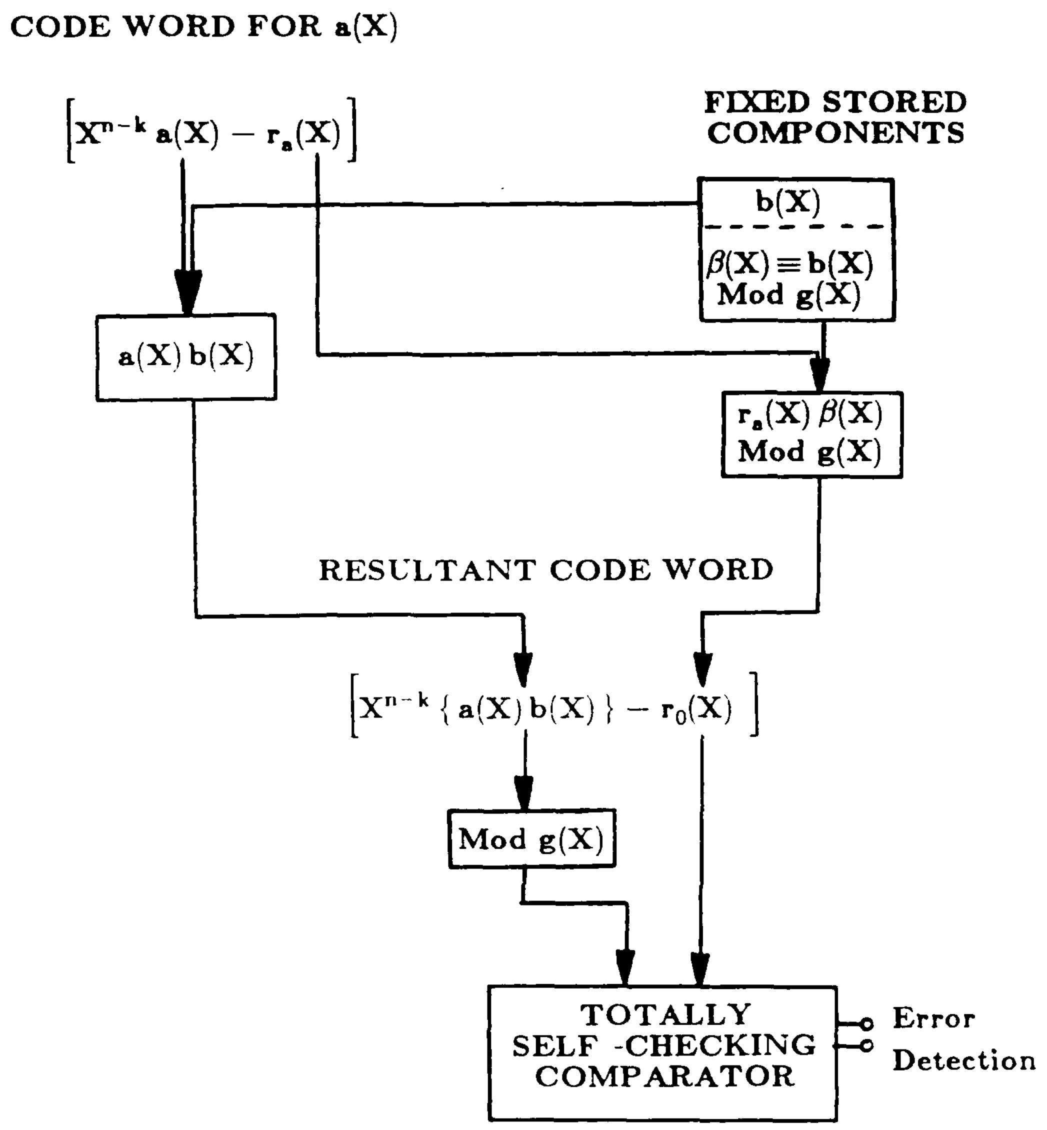
$$\beta(X)r_a(X) = q_0(X)g(X) + r_0(X) \qquad (13)$$

After cancellation of terms, the code representation for [a(X)b(X)] is clearly a code word.

$$X^{n-k}[a(X)b(X)] - r_0(X) \equiv g(X)[q_a(X) + q_0(X)] \bmod (X^n-1)$$

Thus this abbreviated method still produces a code word, with all the error detecting potential of the code, but with a simpler formula for the parity portion.

This reduced special case occurs because only one of the parity parts $r_a(X)$ emanates from $X^{n-k}a(X)$, allowing ease in separating the effects of data and parity. The slightly less complex implementation of this method is depicted in Figure 6. The residue of the fixed sequence $\beta(X)$ is stored as well to expedite the parity formation.


CODE WORD FOR a(X)
$\left[X^{n-k} a(X) - r_a(X)\right]$
FIXED STORED COMPONENTS
b(X)
$\beta(X) \equiv b(X)$ Mod g(X)
a(X) b(X)
$r_a(X)\,\beta(X)$ Mod g(X)
RESULTANT CODE WORD
$\left[X^{n-k}\{a(X)\,b(X)\} - r_0(X)\right]$
Mod g(X)
TOTALLY SELF -CHECKING COMPARATOR
Error Detection

PROTECTING CONVOLUTION WHEN ONE SEQUENCE IS KNOWN
FIGURE 6

Another situation of interest is cyclic convolution which is sometimes a byproduct of the implementation methods. For example, Fourier transforms used to form a convolution intrinsically produces cyclic convolution [2,16]. Cyclic convolution treats two sequences as periodically extended so that any components developed during the convolution process lying beyond the basic length are effectively wrapped around. The polynomial equivalent of length k cyclic convolution is described through a modulo reduction using factor (X^k-1).

$$a(x)b(x) \equiv d(x) \bmod (X^k-1)$$

The cyclic convolution is the remainder part in the Eucledian Algorithm while the quotient, t(x), will be important shortly.

$$a(x)b(x) = t(x)(x^k-1) + d(x) \qquad ; \quad \deg d(x) < k \tag{14}$$

The product a(x)b(x) represents the linear convolution which is modified by shifted versions of t(x) yielding the cyclic convolution d(x).

$$d(x) = a(x)b(x) - x^k t(x) + t(x)$$

The shifted terms in $x^k t(x)$ are subtracted from the linear convolution eliminating terms beyond X^k whereas t(x) is added into the lower terms, the end-around effects.

The parity positions corresponding to the cyclic convolution is $r_d(x)$ defined in an equation reminiscent of equation (6).

$$X^{n-k}d(x) = q_d(x)g(x) + r_d(x) \quad \bmod (X^n-1) \tag{15}$$

Previously given expressions for a(x) and b(x) involving $r_a(x)$ and $r_b(x)$ respectively may be substituted into equation (14), expressly showing the dependency of c(x) on these parity parts. Equation (15) then relates there original parity terms to the desired parity part $r_d(x)$ where several simplifications are possible particularly because fo the modulo (X^n-1) reduction.

$$\begin{aligned} &r_d(x) - (X^{n-k}-1)t(x) - X^k r_a(x)r_b(x) \equiv \\ &\quad \{X^k[q_a(x)q_b(x)g(x) + r_a(x)q_b(x) + r_b(x)q_a(x)] - q_d(x)\}g(x) \end{aligned} \tag{16}$$

However g(x) divides (X^n-1), by the code's construction, permitting an identity modulo g(x).

$$r_d(x) \equiv (X^{n-k}-1)t(x) + X^k r_a(x) r_b(x) \mod g(x) \tag{17}$$

The new parity related with cyclic convolution d(x) contains the weighted and reduced product of the inputs' parities, as in equation (8a), and a component involving the end-around modification t(x) inherent in cyclic convolution. Its weighting factor is e(x).

$$r_d(x) \equiv e(x)t(x) + f(X) r_a(x) r_b(x) \mod g(x) \tag{18a}$$

$$e(x) \equiv (X^{n-k}-1) \mod g(x) \tag{18b}$$

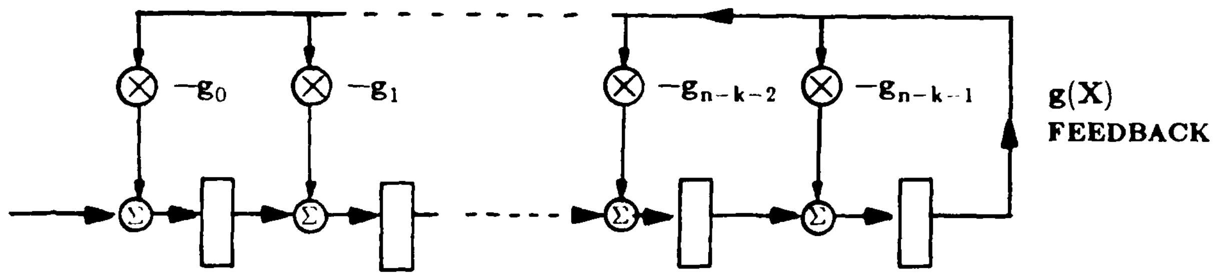
Thus the cyclic convolution parity requires knowledge of the end-around terms, either directly or through a modulo g(x) reduced version.

COMPLEXITY OF PARITY GENERATION

There are numerous ways to realize the convolution and modulo reductions prescribed in the previously described methods. They range from distributed arithmetic processors to time-multiplexing a high-speed ALU resource. In order to study the general complexity, realizations as shown in Figure 7 are considered. Both the parity calculation and recalculation operations involve reduction modulo g(X). For example, the (n-k) parity positions represent the remainder after g(X) is divided into a shifted version of the information data, equation (6). Figure 7a shows one viewpoint of the process using a feedback configuration. The k data samples are inserted serially and the remainder is developed by the feedback paths effecting the polynomial division.

On the other hand, a slight modification of the basic feedback configuration computes the product $r_a(X)r_b(X)$ modulo g(X). Figure 7b depicts such a system where the parity values in $r_a(X)$ represent the inputs. Furthermore the output from this system can be passed through a similar configuration where the input scaling taps are defined by f(x) to produce the parity calculation dictated by equation (8a).

These basic principles are central to all other configures including a wide range of distributed and parallel computational schemes. Nevertheless the polynomial reduction process is essentially sequential, and the viewpoints in Figure 7 are typical of the complexity. There are (n-k) storage locations in Figure 7 and the leading coefficient of the code generating polynomial g(X) is taken as 1, without loss of generality. The lower portion of this figure implements the product $r_a(X)r_b(X)$, while the feedback path in the upper part performs and mod g(X) reduction simultaneously. As noted earlier, in real cyclic codes the



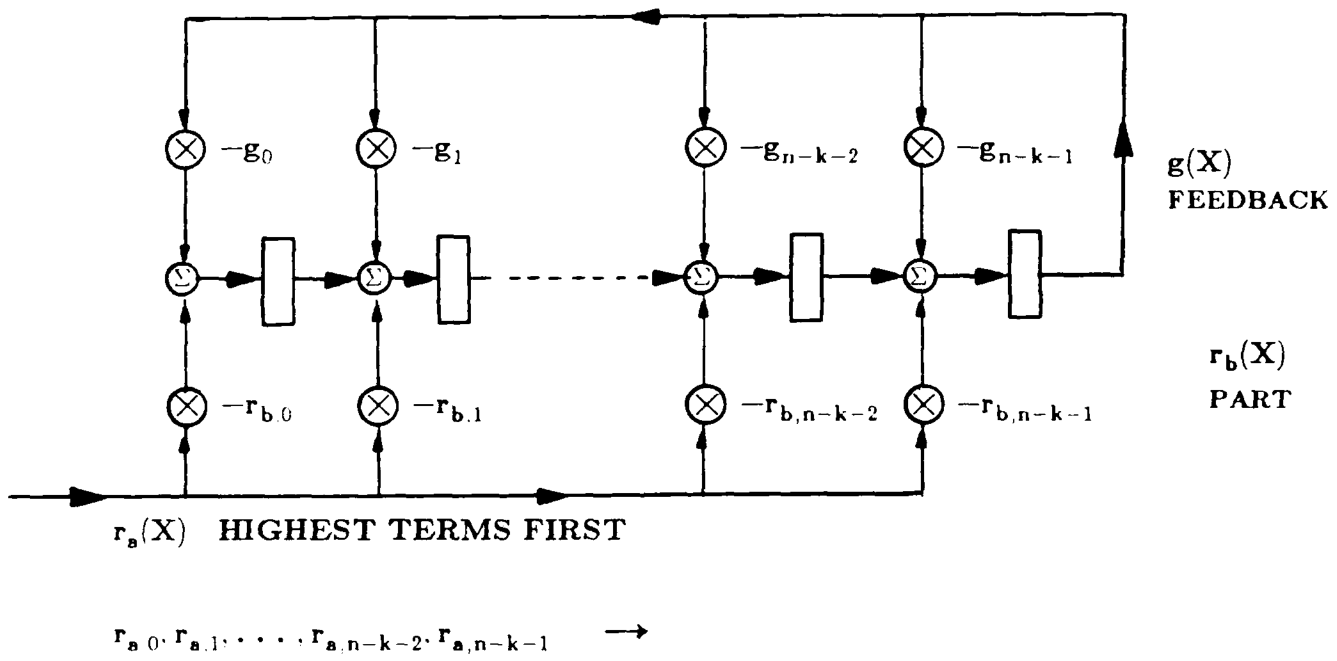
Input
a(X) HIGHEST TERMS FIRST

$a_0, a_1, \ldots, a_{k-2}, a_{k-1} \longrightarrow$

Remainder Is Contents
After Last Position Entered
HIGHEST ORDER TO THE RIGHT

Part Of Parity Generation
$r_a(X) \equiv \{X^{n-k}\, a(X)\}$ Mod g(X)
Figure 7a.

PARITY GENERATION OR REGENERATION
FIGURE 7



Answer Remains In Registers,
HIGHEST ORDER TO THE RIGHT

Part Of Parity Generation
$-\{r_a(X)\, r_b(X)\}$ Mod $g(X)$
Figure 7b.

PARITY GENERATION OR REGENERATION
FIGURE 7

coefficients of g(X) have symmetric or anti-symmetric properties about the midpoint. The number of multiplications in the feedback path can be halved approximately. This property, however, will not be assumed in the complexity estimates below.

One complexity measure may be a count of the number of multiplications and additions required. The product part needs multiplications and additions on the order of the following estimates.

$$\text{PRODUCT PART} \begin{cases} \text{MULTIPLICATIONS:} & (n-k)(n-k+1) \\ \text{ADDITIONS;} & (n-k)(n-k-1) \end{cases}$$

On the other hand, the modulo reduction is heavily influenced not only by the degree of g(X), but also by the number of nonzero coefficients. The notation $|g|$ will denote the number of nonzero terms in g(X), including the leading coefficient presently assumed to be 1. Then the additional number of multiplications and additions for the feedback part may be estimated.

$$\begin{array}{l}\text{MODULO}\\ \text{REDUCTION}\\ \text{PART}\end{array} \begin{cases} \text{MULTIPLICATIONS:} & (n-k-1)(|g|-1) \\ \text{ADDITIONS:} & (n-k-1)(|g|-1) \end{cases}$$

These estimates will be helpful in projecting the overall complexities. The are two different approaches that may be taken in realizing the parity calculations dictated by equation (8a). One approach would first form the triple product $\{f(X)r_a(X)r_b(X)]$, yielding a polynomial with possible highest degree 3(n-k-1), followed by the modulo reduction. This approach uses more interim memory locations. An alternate method would interconnect two configurations of the type in Figure 7b. With this realization equation (8a) is implemented in two stages (see Figure 4b), say first producing

$$\alpha(X) \equiv r_a(X)r_b(X) \bmod g(X) \ ,$$

and then completing the calculations with

$$r_{ab}(X) \equiv \alpha(X)f(X) \bmod g(X) \ .$$

Straightforward estimates show that either approach employs on the same respective orders of multiplications and additions.

$$\text{ORDER OF PARITY GENERATION COMPLEXITY} \begin{cases} \text{MULTIPLICATIONS:} & 2[(n\text{-}k)^2 + (n\text{-}k\text{-}1)(|g|\text{-}1)] \\ \text{ADDITIONS:} & 2(n\text{-}k\text{-}1)[(n\text{-}k) + (|g|\text{-}1)] \end{cases}$$

The dominant factor in both items is $(n\text{-}k)^2$, a quantity related to the number of parity positions in the code, and the error-detecting ability of the system.

SUMMARY

Recently proposed generalized cyclic codes defined over the common arithmetic structures usually employed in practical data processing implementations dictate the parity samples for protecting convolution operations. The parity positions associated with each input array are convolved along with a simply defined, comparably sized, stored weighting array. A wide range of quite powerful generalized cyclic codes are available with protection levels proportional to the number of parity positions. The new systematic encoding and processing techniques make the system realization straightforward and efficient. The protection effort requires additional arithmetic complexity on the order of the number of parity positions squared.

REFERENCES

1. J. H. McClellan and C. M. Rader, Number Theory in Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979.

2. H. J. Nusbaumer, Fast Fourier Transform and Convolution Algorithms. New York: Springer-Verlag, 1981.

3. R. E. Blahut, Fast Algorithms for Digital Signal Processing. Reading, Massachusetts: Addison-Wesley Publishing Co., 1985.

4. W. W. Peterson and E. J. Weldon, Jr., Error-Correcting Codes, Second Edition. Cambridge, Massachusetts: The MIT Press, 1972.

5. P. Shankar, "On BCH Codes Over Arbitrary Integer Rings," IEEE Transactions on Information Theory, Vol. IT-25, pp. 480-483, July 1979.

6. T. G. Marshall, Jr., "Coding of Real-Number Sequences for Error Correction: A Digital Signal Processing Problem," IEEE Journal of Selected Areas in Communications, Vol. SAC-2, pp. 381-392, March 1984.

7. S. Lin and D. J. Costello, Jr., Error Control Coding: Fundamentals and Applications. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

8. N. Jacobson, Basic Algebra I. San Francisco: W. H. Freeman and Co., 1974.

9. I. F. Blake, "Codes Over Certain Rings," Information and Control, Vol. 20, pp. 396-404, 1972.

10. I. F. Blake, "Codes Over Integer Residue Rings," Information and Control, Vol. 29, pp. 295-300, 1975.

11. E. Spiegel, "Codes Over Z_m," Information and Control, Vol. 35, pp. 48-51, 1977.

12. E. Spiegel, "Codes Over Z_m, Revisited," Information and Control, Vol. 37, pp. 100-104, 1978.

13. J. Wakerly, Error Detecting Codes, Self-Checking Circuits and Applications. New York: North-Holland, 1978.

14. J. K. Wolf, "Redundancy, the Discrete Fourier Transform, and Impulse Noise Cancellation," IEEE Transactions on Communications, Vol. COM-31, pp. 458-461, March 1983.

15. J. K. Wolf, "Redundancy and the Discrete Fourier Transform," Proceedings of the 1982 Conference on Information Sciences and Systems, pp. 156-158, 1982.

16. N. K. Bose, Digital Filters Theory and Applications. New York: North-Holland, 1985.

APPENDIX B

PROTECTING IIR FILTER REALIZATIONS WITH EMBEDDED CONVOLUTIONAL CODES

INTRODUCTION

Digital filter implementations need to be protected against both hard and soft failures emanating from their underlying electronic realizations. The level of protection may range from the detection of erroneous behavior to the correction of faulty outputs, a situation sometimes referred to as error masking [1,2]. Many protection mechanisms rely on placing redundancy, either internal or external, in the normal system operation, and structured linear block or convolutional codes [3,4] represent controlled redundancy which can be used for this purpose. Such codes have their early foundation in communications systems and are generally defined over finite algebraic structures because of the nature of digital communications. However, newer classes of codes have been developed with symbols from the real or complex number fields [5], the same algebraic setting for digital filter realizations.

This paper presents a new method for protecting digital filters by employing the error-detecting distance properties of real convolutional codes. The normal filter system will be augmented with adjacent parallel parity calculations whose values are compared against parity quantities recalculated from the normal system's output. Certain classes of errors occurring anywhere in the overall system including the parity calculation and recalculation subsystems are easily detected. Furthermore in this way the speed performance of the original filter's operations is not adversely affected. A convolutional code is ideally suited to the fundamentally infinite processing involved in Infinite Impulse Response (IIR) filter designs. Because of the inherent feedback within this type of filter realization, errors can propagate and increase ultimately overwhelming the protecting distance of the code. This is particularly true for block codes where framing a large number of errors in one block can easily exceed the code's detection capability. On the other hand, a convolutional code with its continuously increasing memory can sense the onset of errors before they increase beyond detection limits.

A systematic real convolutional code [3,5] of rate $\frac{k}{n}$ is applied externally to a digital filter, producing (n-k) parity samples for every group of k data samples processed in the normal filter. A corresponding group of (n-k) parity values is recalculated from the filter's output samples and compared against the group calculated in parallel. Each group is produced by parity filter channels operating at a rate decimated by k [6].

Significant savings in the parity channels' complexity are achieved by modifying the real convolutional code, with minimal impact on its error-detecting capability, thus greatly simplifying their implementations. Equivalent performance is possible with only Finite Impulse Response (FIR) filter structures in these parity channels, permitting the decimation to commute with the filters themselves [6].

The theory and practice supporting the code modification techniques are fully developed. An example demonstrates the general method. A statistical analysis of the error detection system guides the selection of thresholds at the parity comparison unit, and bounds on the comparator's output noise are established. An alternative method using block codes involving both internal states and the output at every filter sample is analyzed. This state variable realization may be viewed externally and compared with the new convolutional code approach showing the latter's greater flexibility and efficiency.

The next section develops the theoretical basis for embedding real convolutional codes around a digital filter system without degrading the speed performance. The necessary parity channel filters are defined and described. The following section examines modifying the convolution code without loss in error protecting performance while simultaneously simplifying the parity channels' realizations. A practical approach to code modification is presented where a simple example is detailed. A statistical analysis of the parity comparator's output noise is developed and a bound on its variance is given. In the final section an alternate method which uses block codes and a state-variable realization is examined.

CONVOLUTIONAL CODES IN FILTER REALIZATIONS

General digital filters and IIR filters in particular may be expressed in several equivalent ways with some viewpoints more similar to convolutional code structures than others. The transfer function (pulse transfer function) [7-9] is formally related to a filter's difference equation through the Z transform.

DIFFERENCE EQUATION

$$v(k)=\sum_{i=0}^{P} a_i u(k-i) - \sum_{j=1}^{Q} b_j v(k-j) \quad ; \tag{1}$$

$u(i)$ *Input Sequence*
$v(k)$ *Output Sequence*
$i,k=0,1,2,.....$

The input and output samples, u(i) and v(i) respectively, are undefined for negative index values (or assumed to be zero).

TRANSFER FUNCTION

$$H(Z) = \frac{V(Z)}{U(Z)}, \tag{2}$$

where the components in the ratio are Z transforms of the respective input and output sequences. They are related to the difference equation's governing coefficients.

$$V(Z) = \sum_{i=0}^{+\infty} v(i) Z^{-i} \tag{3a}$$

$$U(Z) = \sum_{i=0}^{+\infty} u(i) Z^{-i} \tag{3b}$$

$$H(Z) = \frac{a_0 + a_1 Z^{-1} + a_2 Z^{-2} + \ldots + a_\nu Z^{-\nu}}{b_0 + b_1 Z^{-1} + b_2 Z^{-2} + \ldots + b_\delta Z^{-\delta}} \qquad ; b_0 = 1 \tag{3c}$$

The impulse response is the inverse Z transform of H(z) and provides another representation of the input/output relationship.

$$H(Z) \longleftrightarrow \{h_j\}_{j=0}^{+\infty}$$

$$v(k) = \sum_{j=0}^{+\infty} h_j \, u(k-j) \qquad ; k = 0,1,2,\ldots \tag{4}$$

Another equivalent expression for an IIR filter employs semi-infinite indexed vectors and matrix.

$\underline{u} = \left[u(0), u(1), u(2), \ldots \right]$ *INPUT ROW VECTOR*

$\underline{v} = \left[v(0), v(1), v(2), \ldots \right]$ *OUTPUT ROW VECTOR*

$$F = \begin{pmatrix} h_0 & h_1 & h_2 & h_3 & h_4 & - & - & - & - \\ 0 & h_0 & h_1 & h_2 & h_3 & - & - & - & - \\ 0 & 0 & h_0 & h_1 & h_2 & - & - & - & - \\ 0 & 0 & 0 & h_0 & h_1 & - & - & - & - \\ 0 & 0 & 0 & 0 & h_0 & - & - & - & - \\ 0 & 0 & 0 & 0 & 0 & - & - & - & - \\ | & 0 & 0 & 0 & 0 & 0 & | & | & | \\ | & | & | & | & | & | & | & | & | \end{pmatrix}$$

$$\underline{v} = \underline{u}F \tag{5}$$

Convolutional codes have a similar defining structure. Although traditionally defined over finite field alphabets [3], recent research results show how they may be extended to systems using either integer or real arithmetic [5]. Nevertheless, the basic approach to convolutional codes remain the same, particularly with regard to a matrix description of the encoding and parity checking functions. Only systematic forms of convolutional codes, a situation where the information positions are clearly identified in the encoded output, will be considered primarily because they are well-suited to checking the filtering operation and are automatically noncatastrophic [1] [3].

The encoding matrix for a systematic convolutional code, G, has a block-type format involving m fundamental finite sized matrices whose dimensions are related to the rate and number of parity check positions in the code. The parameter m determines the constraint length of the code.

[1] A catastrophic convolutional code permits a finite number of errors to produce a valid code sequence implying a confusion with an infinite number of errors.

$$G = \begin{vmatrix} G_0 & G_1 & - & - & G_m & 0 & - \\ 0 & G_0 & - & - & G_{m-1} & G_m & - \\ 0 & 0 & - & & - & G_{m-1} & - \\ 0 & 0 & \vert & \vert & & - & \vert \\ 0 & 0 & & & - & & \\ \vert & \vert & \vert & & G_1 & - & \vert \\ & & & \vert & G_0 & G_1 & \\ \vert & \vert & \vert & & 0 & G_0 & \vert \\ & & & & 0 & 0 & \\ \vert & \vert & \vert & \vert & \vert & \vert & \vert \end{vmatrix} \tag{6}$$

Each submatrix G_i is $k \times n$ where the ratio $\frac{k}{n}$ is called the rate of the code and (n-k) is the number of parity positions provided for every k input digits. For the systematic case, matrix G_0 has a particularly distinctive form.

$$G_0 = \left| I \mid P_0 \right| \qquad ; I \;\; k \times k \;\; \textit{Identity Matrix} \tag{7a}$$
$$\qquad\qquad P_0 \;\; k \times (n-k) \;\; \textit{Parity-Check Matrix}$$

$$G_j = \left| \mathbf{0} \mid P_j \right| \qquad ; \mathbf{0} \;\; k \times k \;\; \textit{Zero Matrix} \tag{7b}$$
$$\qquad\qquad P_j \;\; k \times (n-k) \;\; \textit{Parity-Check Matrix}$$

$$j = 1, 2, \ldots, m.$$

The entries in the parity-check submatrices, P_j are either 0 or 1 even for the real arithmetic case [5]. Encoding the input data stream, represented by infinite row vector $\underline{x}$, produces an infinite output row vector $\underline{w}$ with (n-k) parity-check positions interleaved as a subblock between every block of k input values.

$$\underline{w} = \underline{v}G \tag{8}$$

$$\underline{w} = \Big\{ w(0), w(1), \ldots, w(k-1), \underbrace{w(k), w(k+1), \ldots, w(n-1)}_{Parity\ Positions}, w(n),$$

$$w(n+1), \ldots, w(n+k-1), \underbrace{w(n+k), w(n+k+1), \ldots, w(2n-1)}_{Parity\ Positions}, w(2n), w(2n+1), \ldots$$

$$\ldots\ \underbrace{w(sn+k), w(sn+k+1), \ldots, w((s+1)n-1)}_{Parity\ Positions}, \ldots \Big\}$$

The parity positions interspersed between groups of k input digits are an effect of the stacks of the (m+1) matrices G_0, $G_1, \ldots, G_m$ in the block-shifted format evident in G. In particular, the $(k \times k)$ identity matrix, I, part of matrix G_0, copies the input data from $\underline{v}$ directly through to the identical groups in $\underline{w}$;

$$w(sn+j) = v(sk+j) \qquad ; s=0,1,2,\ldots\ldots \quad j=0,1,2,\ldots,(k-1). \tag{9}$$

The parity positions are a function of possibly (m+1)k input samples through the action of the P_j parts of each G_j. The stack of these parity weighting values will be denoted by an $\{(m+1)k \times (n-k)\}$ matrix Q with respective columns $\{\underline{q_r}\}$

$$Q = \begin{vmatrix} P_m \\ P_{m-1} \\ | \\ | \\ | \\ | \\ P_2 \\ P_1 \\ P_0 \end{vmatrix} = \Big(\underline{q_0}, \underline{q_1}, \underline{q_2}, \ldots\ldots, \underline{q_{n-k-1}} \Big). \tag{10a}$$

$$\underline{q_c} = ((q_c^{(j)})) \qquad ; j=0,1,2,\ldots,(m+1)k-1. \qquad (10b)$$
$$\underline{q_c} \ (m+1)k\times 1 \ \textit{Column Vector}$$
$$c=0,1,2,\ldots,(n-k-1).$$

Consequently generic parity-check position w(sn+k+r) is obtained by the weighting action of column $\underline{q_r}$. Each parity value may be viewed as the output of an FIR filter, described notationally using the Z transform of column $\underline{q_c}$.

FIR FILTER EFFECT, COLUMN c.

$$Q_c(Z) = \sum_{j=0}^{(m+1)k-1} q_c^{((m+1)k-1-j)} Z^{-j} \qquad ; c=0,1,2,\ldots,(n-k-1). \qquad (11)$$

When a convolutional code is used to encode the output of an IIR filter such as the one described by matrix F, equation (5), the combined effect in terms of matrices becomes:

$$\underline{w} = \underline{u}FG \qquad (12)$$

The properly filtered data still appear in the information positions while the parity positions represent the additional filter weighting introduced by the respective columns of Q, equations (10). The parity sample is produced for every k input data samples corresponding to a decimated filter at rate $\frac{1}{k}$ [6]. Each output of a parity channel represents the combined effects of the original filter coupled with the respective FIR filter defined by the $Q_c(Z)$ transfer function; c=0,1,2,. . . . , (n-k-1). This system dichotomy is shown in Figure 1 where IIR filter matrix F is denoted by its equivalent transfer function H(Z) and the decimation action is performed by the decimators, $\downarrow$ k. Fault-tolerance is introduced by using a detection system, as depicted in Figure 2 where the parity channel output values are compared against associated quantities calculated by similar parity weighting channels operating on the output of filter H(Z). Any discrepancies, within the round-off tolerance of the system and the error-detecting power of the code, indicates a protected malfunction somewhere in the system, including even the subsystems generating or regenerating the parity values.

Of course, this system configuration is grossly inefficient since it replicates (n-k) actions of the original filter, one in each parity channel. Even the decimated sampling at reduced rate $\frac{1}{k}$ does not mitigate the increased complexity. However other steps can drastically reduce this complexity bringing the implementation in the realm of practical applicability.

SIMPLIFICATION IN THE PARITY CHANNELS

One straightforward complexity reduction method follows directly from the nature of the 0 and 1 entries in the FIR channel filter weights. A typical parity channel is the cascade of the original filter, described by its transfer function H(Z), with the code's FIR filter denoted generically by $Q_c(Z)$. (See Figure 2 and equation (11)). On the other hand, the original transfer function is the ratio of two polynomials N(Z) and D(Z).

$$H(Z) = \frac{N(Z)}{D(Z)}$$

$$N(Z) = a_0 + a_1 Z^{-1} + a_2 Z^{-2} + \ldots . + a_\nu Z^{-\nu} \tag{13a}$$

$$\nu \leq \delta$$

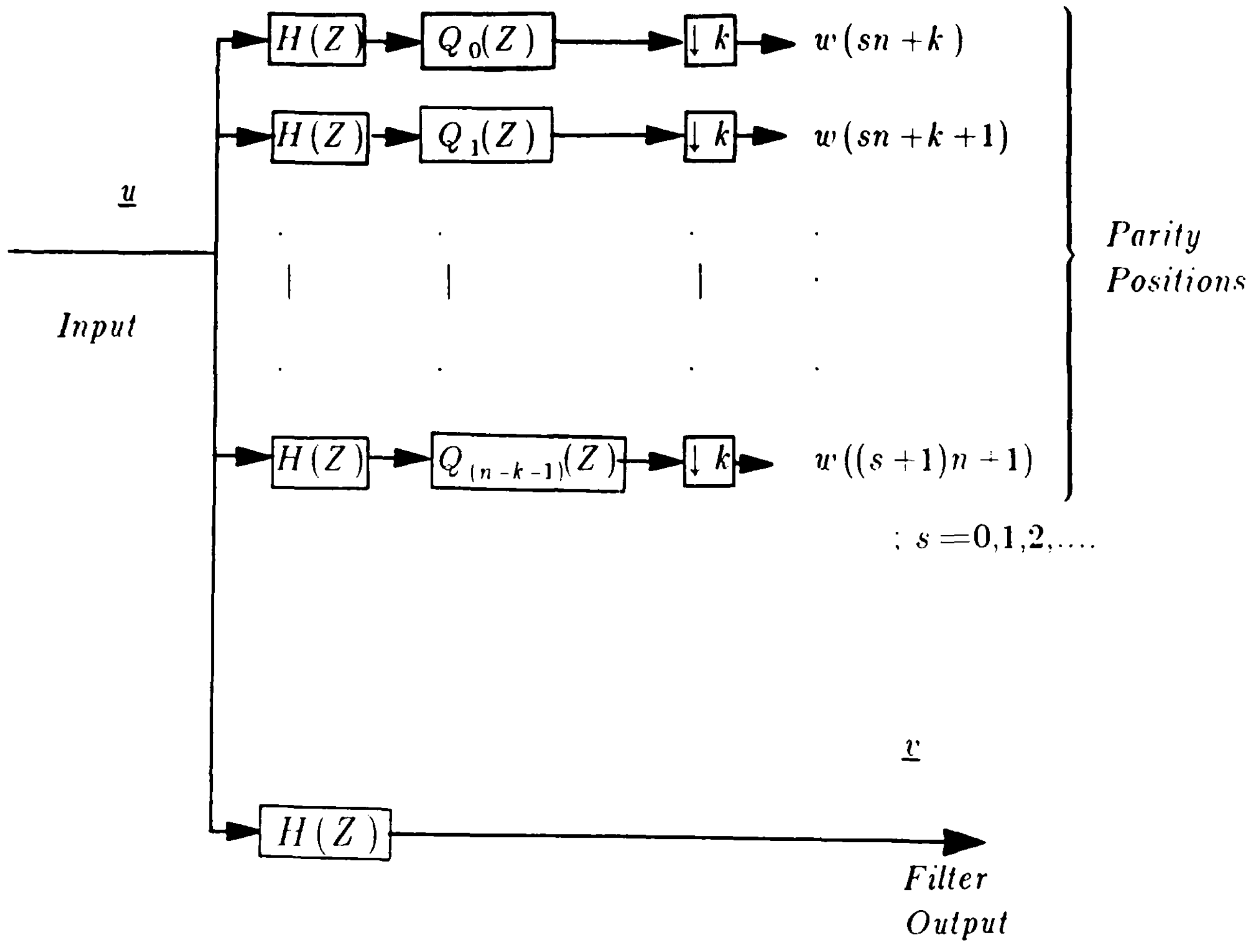
$$D(Z) = 1 + b_1 Z^{-1} + b_2 Z^{-2} + \ldots . + b_\delta Z^{-\delta} \tag{13b}$$

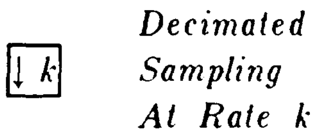
The numerator function N(Z) in each paity channel may be commuted with the FIR weighting $Q_c(Z)$. Since only every k th output sample from the parity channel is required, the multiplications indicated by the numerator function N(Z) are performed only after k samples have been shifted into its implementation. Figure 3 displays a typical interchange for one realization form. However, one disadvantage is the additional memory necessary for the feedback portion of the altered cascade, due to the generally increased length of the parity channel weighting. Normally the code's constraint length, (m+1)k, is larger than the number of filter poles, the degree δ of D(Z). Even though this additional storage is not a dominating factor in modern design, there are still (n-k) separate filters, each with feedback required.

The structure of a convolutional code permits several degrees of freedom for further simplifying the implementation complexity in the (n-k) parity channels. The error control capability of a convolutional code, even in the case of real arithmetic codes [5], depends primarily on the location of the NONZERO entries in the parity parts of the generator matrix [Chapter 10, 3]. This translates into a constraint that the error-protecting distance of a real convolutional code remains the same as long as the zero terms in the parity weighting transfer functions

ENCODED FILTER

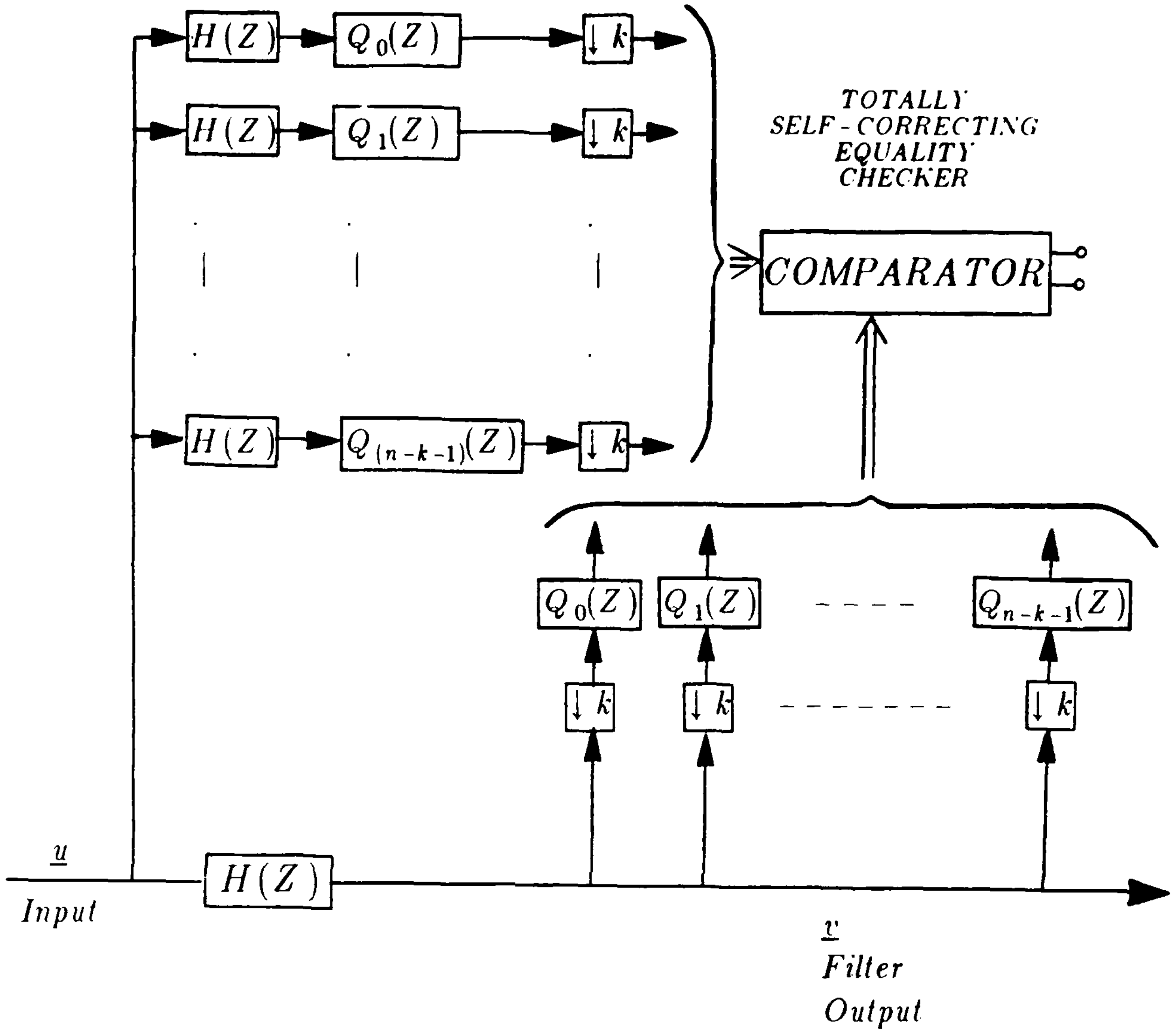
$$\underline{\mathbf{w}} = \underline{\mathbf{u}}\mathbf{FG}$$

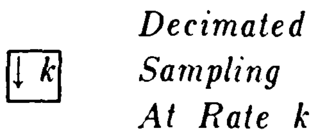




BASIC CONVOLUTIONALLY ENCODED FILTER SYSTEM

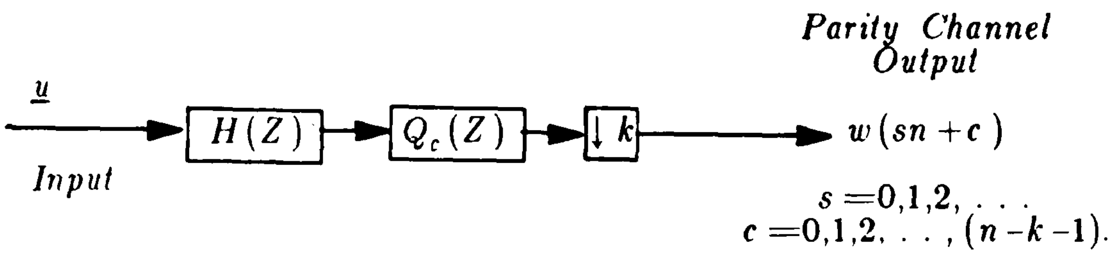
Figure 1





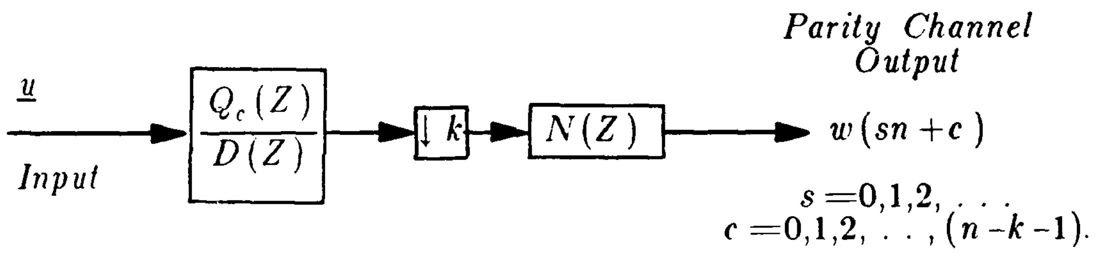
EXTERNALLY PROTECTED FILTER SYSTEM

Figure 2



$$H(Z) = \frac{N(Z)}{D(Z)}$$

(a.) ORIGINAL CASCADE IN PARITY CHANNEL



(b.) INTERCHANGED NUMERATOR FUNCTIONS

$Q_c(Z)$ HAS ONLY ZERO OR UNITY COEFFICIENTS

SIMPLIFICATION BY INTERCHANGING NUMERATOR FUNCTIONS

Figure 3

$Q_0(Z)$, $Q_1(Z)$, . . . , $Q_{n-k-1}(Z)$, are preserved while permitting the nonzero values to change, within acceptable levels considering the effects of roundoff accuracies. Nevertheless, all signs of nonzero entries in any row of **Q** must be the same in its modified form. The impact and virtue of altering the parity weighting filters is developed considering a generic parity channel transfer function $Q_c(Z)$.

$$Q_c(Z) = q_c^{(M-1)} + q_c^{(M-2)} Z^{-1} + \ldots + q_c^{(0)} Z^{-(M-1)} \qquad M=(m+1)k \tag{14}$$

The parity channel can be simplified greatly if an equivalent convolutional code can be constructed so as to eliminate the poles in the original filter. Label the new equivalent parity weighting transfer function by $Q_c'(Z)$, and then the desired goals are to select another polynomial $R_c(Z)$ such that

$$D(Z)\,R_c(Z) = Q_c'(Z), \tag{15a}$$

AND

$$\left\{\begin{matrix} SET\ OF\ ZERO \\ COEFFICIENTS \\ IN\ Q_c(Z) \end{matrix}\right\} = \left\{\begin{matrix} SET\ OF\ ZERO \\ COEFFICIENTS \\ IN\ Q_c'(Z) \end{matrix}\right\}. \tag{15b}$$

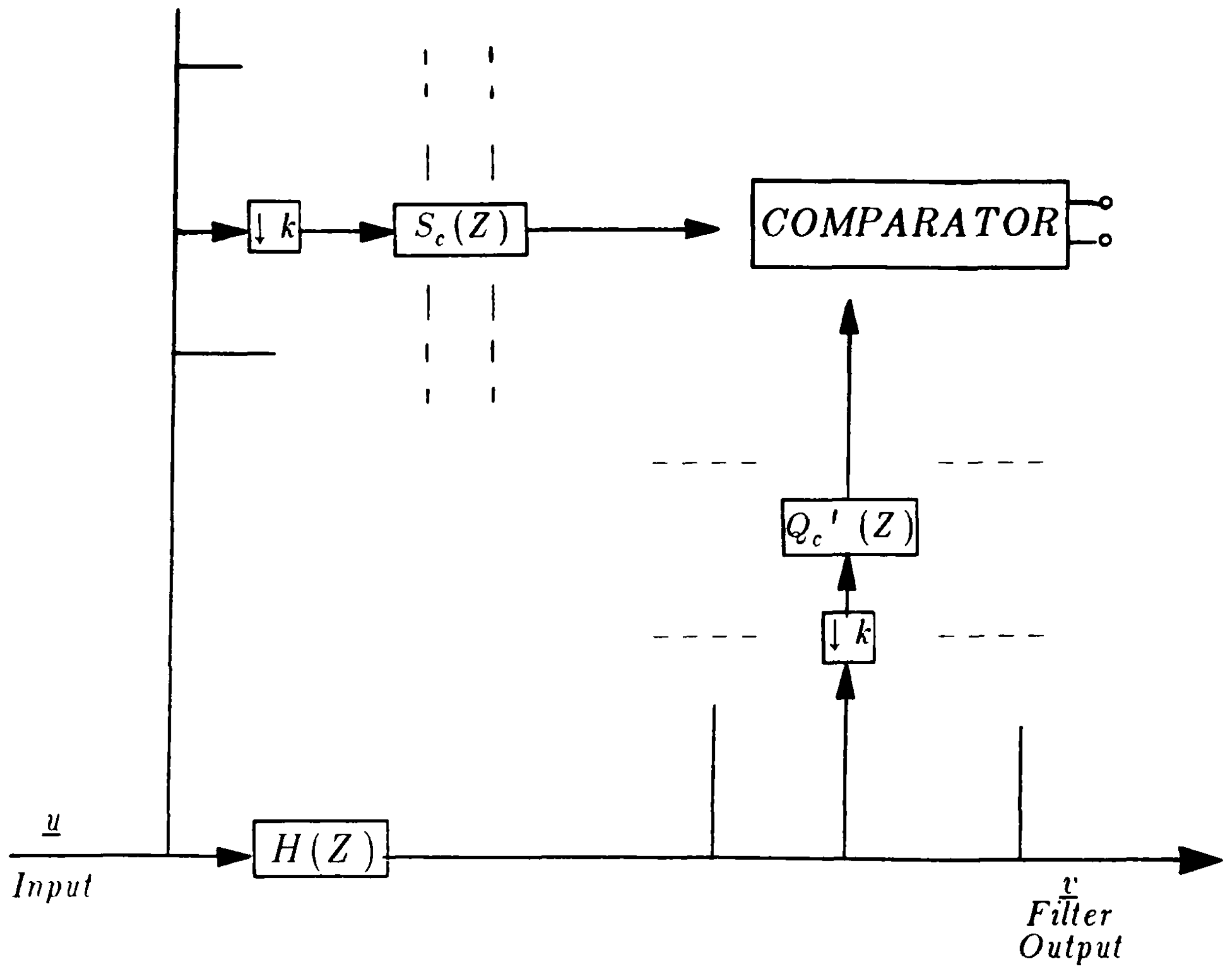
Leaving aside for the moment questions concerning the existence and relative sizes of the nonzero coefficients in this new $Q_c'(Z)$, the impact of such a change, at least theoretically is to remove the feedback portion in each parity channel.

$$H(Z)Q_c'(Z) = R_c(Z)N(Z) \equiv S_c(Z) \tag{16}$$

The parity channel only needs to implement the FIR filter described by the transfer function $S_c(Z)$. In addition, the decimation operation, $\downarrow$ k, commutes with such filter structures [6]. Figure 4 shows the theoretical impact of this simplification for typical channel c ; c=0,1,. . . . , n-k-1.

DESIGN CONSIDERATIONS

The code modification process has the potential for great simplification in the practical realization of protected digital filters. The general situation can be further constrained to permit only real polynomials in the modified code design, equations (15). A typical equation may be translated into matrix equations with real coefficients.



CODE MODIFICATION SIMPLIFICATION FOR PARITY CHANNEL
Figure 4

$$A\,\underline{r} = \underline{q'} \tag{17a}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ b_1 & 1 & 0 & & \\ b_2 & b_1 & | & | & | \\ b_3 & b_2 & & & \\ - & - & - & 0 & 0 \\ - & - & & 1 & 0 \\ - & - & - & b_1 & 1 \\ b_\delta & b_{\delta-1} & & b_2 & b_1 \\ 0 & b_\delta & | & | & b_2 \\ 0 & 0 & & & | \\ | & | & | & b_\delta & \\ 0 & 0 & 0 & 0 & b_\delta \end{pmatrix} \times \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ - \\ - \\ . \\ | \\ . \\ r_{M-\delta-1} \end{pmatrix} = \begin{pmatrix} q_0' \\ q_1' \\ q_2' \\ - \\ - \\ . \\ | \\ . \\ q_{M-1}' \end{pmatrix} \tag{17b}$$

The $M \times (M-\delta)$ matrix A represents the polynomial D(Z) which when multiplied by R(Z), corresponding to the (M-δ) - rowed column vector $\underline{r}$, yields a modified code polynomial $Q'(Z)$, denoted by $\underline{q'}$.

The row by column products which produce zeros common to both $\underline{q}$ and $\underline{q'}$ are important since they guarantee preserved distance structure in the real convolutional code. Let Ξ be the set of indices labeling the common zero coefficient locations in $\underline{Q}$ and $\underline{Q'}$. Assume that there are ξ items in this set. A homogeneous system can be constructed using only the rows of A with indices in Ξ . The selected rows will be collected into matrix $\tilde{A}$.

$$\tilde{A}\,\underline{r} = \underline{0} \tag{18}$$

This homogeneous system has nontrivial solutions depending on the relationship $\xi < (M - \delta)$. Furthermore, if ρ is the rank of $\tilde{A}$ (with $\rho \leq \xi$), there are exactly $(M - \delta - \rho)$ linearly independent solutions. These spanning solutions may be found by several standard methods. One approach changes $\tilde{A}$ into an upper echelon matrix, $\tilde{A}_{ech}$, a matrix which has zeros in the lower left triangular part.

$$\tilde{A}_{ech}\, \underline{r} = \underline{0} \tag{19a}$$

$$\tilde{A}_{ech} = \left(\tilde{A}_{sq} \mid \tilde{B} \right) \qquad ; \tilde{A}_{sq}\ ,\ \rho \times \rho\ \textit{Upper Triangular} \\ B\ ,\ \rho \times (M-\delta-\rho)\ \textit{Remaining Columns}$$

The echelon matrix may be reconfigured in turn since there are $(M- \delta - \rho)$ linearly independent solutions, thus dividing a solution vector $\underline{r}$ into dependent and independent parts labeled as follows:

$$\underline{r} = \begin{array}{|c|l} r_0 & \\ r_1 & \\ - & \\ \cdot & \textit{Dependent Part} \quad \underline{r_d} \\ | & \\ \cdot & \\ r_{\rho-1} & \\ --- & ----------------- \\ s_0 & \\ s_1 & \\ - & \\ \cdot & \textit{Independent Part} \quad \underline{r_i} \\ | & \\ \cdot & \\ s_{M-\delta-\rho-1} & \end{array} \tag{20}$$

Equation (19a) may be rewritten in equivalent form using this format.

$$\tilde{A}_{sq}\, \underline{r_d} = -B\, \underline{r_i} \tag{21}$$

This is easily solved by finding the inverse of $\tilde{A}_{sq}$ which exists because it is upper triangular due to the partitioning in equation (19b).

$$\underline{r_d} = -\tilde{A}_{sq}^{-1} B \, \underline{r_i} \tag{22}$$

Thus for any set of completely nonzero real numbers $s_0, s_1, \ldots, s_{(M-\ell-\rho-1)}$, a solution $\underline{r_{sol}}$ can be easily constructed.

$$\underline{r_{sol}} = \begin{pmatrix} -\tilde{A}_{sq}^{-1} B \, \underline{r_i} \\ ---- \\ \underline{r_i} \end{pmatrix} \tag{23}$$

When this parameterized solution vector is substituted in the original matrix equation (17a), the modified weighting function $\underline{q'}$ can be developed in terms of the independent variables $s_0, s_1, \ldots, s_{(M-\ell-\rho-1)}$.

$$A \, \underline{r_{sol}} = \underline{q'} \tag{24}$$

The mathematical manipulation package MACSYMA [10] is ideally suited for obtaining the above solution preserving the parameterization. The variables may be altered by heuristic rules to choose desirable convolutional code forms. One useful guideline is to keep the relative magnitudes of the nonzero coefficients in $\underline{q'}$ similar. These techniques will be exemplified in the following filter design example.

A simple example, typical of most digital filter designs [pg. 223, 9], has four poles and zeros with transfer function $H_{ex}(Z)$.

$$H_{ex}(Z) = \frac{0.001836(1+z^{-1})^4}{(1-1.49237\,Z^{-1}+0.85011\,Z^{-2})(1-1.56200\,Z^{-1}+0.64780\,Z^{-2})} \tag{25}$$

Its poles lie within the unit circle and are listed as:

$$Poles = \begin{Bmatrix} 0.78101 \pm j0.19457 \\ 0.74618 \pm j0.54159 \end{Bmatrix}$$

In the interests of simplicity, the filter will be protected by a six error-correcting majority logic decodable convolutional code [Chapter 13 , 4]. Its parameters are k=1, n=2 and m=17. The single parity channel weighting filter has nonzero coefficients in positions 0, 2, 7, 13, 16 and 17, completely specifying its transfer function.

$$Q_1(Z) = 1 + Z^{-2} + Z^{-7} + Z^{-13} + Z^{-16} + Z^{-17} \tag{26}$$

For this example, M=18, $\delta = 4$, and the number of zero terms in $Q_1(Z)$, ξ=12. The solution process yields a rank $\rho = 12$, leaving two independent variables s_0 , and s_1 to be assigned. An interactive choice of these parameters allows both to be taken as one and the resulting modified parity channel weighting transfer function is $Q_1'(Z)$.

$$\begin{aligned} Q_1'(Z) = & -2.20292 - 8.44565\, Z^{-2} - 3.00276\, Z^{-7} + 1.01097\, Z^{-13} \\ & - 1.74398\, Z^{-16} + 0.55074\, Z^{-17} \end{aligned} \tag{27}$$

The corresponding divisor polynomial $R_1(Z)$, from equation (15a), becomes:

$$\begin{aligned} R_1(Z) = & -2.20292 - 6.72862\, Z^{-1} - 20.56236\, Z^{-2} - 42.09636\, Z^{-3} \\ & - 64.07129\, Z^{-4} - 77.98817\, Z^{-5} - 78.14788\, Z^{-6} - 66.91630\, Z^{-7} \\ & - 48.82915\, Z^{-8} - 29.29234\, Z^{-9} - 13.01494\, Z^{-10} - 2.78602\, Z^{-11} \\ & + Z^{-12} + Z^{-13} \end{aligned} \tag{28}$$

On the other hand the parity channel weighting polynomial for the modified code, $S_1(Z)$ is determined as:

$$\begin{aligned} S_1(Z) = 0.001836 \{ & -2.20292 - 15.54032\, Z^{-1} - 60.69438\, Z^{-2} \\ & - 173.52920\, Z^{-3} - 384.94830\, Z^{-4} - 675.82958\, Z^{-5} \\ & - 963.47613\, Z^{-6} - 1145.81838\, Z^{-7} - 1161.40560\, Z^{-8} \\ & - 1016.68640\, Z^{-9} - 768.97223\, Z^{-10} - 492.83269\, Z^{-11} \\ & - 254.23223 Z^{-12} - 93.06822 Z^{-13} - 14.15902\, Z^{-14} \\ & + 7.21398\, Z^{-15} + 5\, Z^{-16} + Z^{-17} \} \end{aligned} \tag{29}$$

The code modification introduces coefficient quantization errors in practical implementations [7-9], particularly with regards to the simplified parity channel's error comparator subsystem, typified in Figure 4. Each comparator channel indicates error detection when the difference between the parity channel's output from decimated FIR filter, described by $S_c(Z)$, and that recomputed through transfer function $Q_c'(Z)$ exceeds a threshold. The coefficient quantization errors appearing at each comparator channel are modeled through modified transfer functions representing the two paths from the common input.

$$\begin{aligned} Q_c''(Z) &= Q_c'(Z) + \epsilon_c(Z) \qquad ; Deg\ \epsilon_c(Z) = (M-1) \\ S_c''(Z) &= S_c'(Z) + \eta_c(Z) \qquad ; Deg\ \eta_c(Z) = (M+\nu-\delta-1) \end{aligned} \tag{30}$$

The transfer function $Q_c''(Z)$ represents the practical realization where the quantization error effects, $\epsilon_c(Z)$, are small; $S_c''(Z)$ corresponds similarly to the real implementation of $S_c(Z)$ where small errors represented by $\eta_c(Z)$ are added to each coefficient

The difference at one comparator's output is governed by an overall transfer function, Δ (Z), from the common input. The channel labeling subscript c is dropped for notational simplicity at this point, and the effect of the decimated sampling is denoted by $\downarrow$ k.

$$\Delta(Z) \equiv \Big[H(Z)\epsilon(Z) - \eta(Z) \Big]_{\downarrow k} \tag{31}$$

There is an input component at the error detector's comparator output. The relative magnitude of the input component is important since the error detection mechanism relies on setting a threshold. After straightforward manipulation and simplification, one detector's output samples, d(s), at decimated sample times (tk), t=0,1,. . . , may be given.

$$d(tk) = \left\{ \sum_{s=0}^{+\infty} h_s'\, u(tk-s) - \sum_{r=0}^{M+\nu-\delta-1} \eta_r\, u(tk-r) \right\} \tag{32}$$

$$; \ t = 0,1,2,\ldots\ldots$$

The first summation represents the cascaded filter H(Z) ϵ (Z) , with impulse response values h_s' , while the second summation corresponds with the filter η(Z). Since both ϵ(Z) and η(Z) have relatively small coefficients, the comparator's output samples are differences of small values.

A statistical analysis approach which leads to a better understanding of the necessary detector thresholds assumes that the input samples represent an uncorrelated input with zero mean and variance σ^2. Thus the input samples u(i) obey the following statistical properties where E{ } is the expectation operator.

$$E\{u(i)\,u(j)\} = \begin{cases} \sigma^2 & i=j \\ 0 & i \neq j \end{cases} \tag{33}$$

The autocorrelation function of one comparator's output, $\phi_{dd}(\tau k)$, is easily formulated using standard techniques [7,8]. The result is nonzero only for zero values of the offset τ.

$$\phi_{dd}(\tau k) = E\{d(tk)\,d((t+\tau)k)\} \qquad ; \tau=0,1,2,\ldots.. \tag{34a}$$

$$\phi_{dd}(\tau k) = \begin{cases} \sigma^2 \left[\sum_{s=0}^{+\infty} (h_s{}')^2 + \sum_{i=0}^{M+\nu-\delta-1} \eta_i{}^2 - 2 \sum_{j=0}^{M+\nu-\delta-1} h_j{}'\,\eta_j \right] & ; \tau=0 \\ 0 & ; \tau \neq 0 \end{cases} \tag{34b}$$

An upper bound on the variance of the comparator's output samples contains the impulse response coefficients of both branches from the input to the comparator, Figure 4.

$$\phi_{dd}(0) \leq \sigma^2 \left\{ \left[\sum_{s=0}^{+\infty} (h_s{}')^2 \right]^{\frac{1}{2}} + \left[\sum_{i=0}^{M+\nu-\delta-1} (\eta_i)^2 \right]^{\frac{1}{2}} \right\}^2 \tag{35}$$

There are well-known techniques for numerically calculating the summed square values of the modified impulse response $\{h_s{}'\}$. It was first developed by Dugre, Beex and Scharf [11] and is completely detailed, including a computer program in Chapter 5 [7].

A COMPETING METHOD

Another method proposed in the literature [12] applies specifically to a state-variable based realization of a digital filter. The next state and output mappings are encoded with a suitable real block code at each computational step. This protection technique will be contrasted with the convolutional code approach just developed. The internal effect of the block code must be reflected to the output in order to see the implications of the coded state-variable method.

A digital filter implemented in state-variable form may be described through the linear mappings of the next state and the output equations. The state size will be denoted by δ.

NEXT STATE EQUATION
$$x(j+1) = A\,x(j) + B\,u(j) \qquad (36a)$$
$$j=0,1,2,.....$$

OUTPUT EQUATION
$$v(j) = C\,x(j) + d\,u(j) \qquad (36b)$$

The labeling and respective sizes of the vectors and matrices are summarized below.

INPUT SAMPLE	$u(j)$	
OUTPUT SAMPLE	$v(j)$	
STATE VECTOR	$x(j)$	$(\delta \times 1)$
STATE MATRIX	A	$(\delta \times \delta)$
INPUT MATRIX	B	$(\delta \times 1)$
OUTPUT MATRIX	C	$(1 \times \delta)$
INPUT WEIGHTING	d	(1×1)

The complete system description may be combined into one matrix equation involving partitioned vectors and a matrix. An aggregate vector constructed from the next state vector $x(j+1)$ and the present output v(j) is a linear function of another partitioned vector containing the present state $x(j)$ and present input u(j).

$$\begin{pmatrix} x(j+1) \\ v(j) \end{pmatrix} = \begin{pmatrix} A & B \\ C & d \end{pmatrix} \begin{pmatrix} x(j) \\ u(j) \end{pmatrix} \qquad (37)$$

The filter's internal operation is protected by encoding the $(\delta+1)$ components in the aggregate vector. The code is defined through a parity-check matrix H establishing t parity values to be appended as additional rows on the left side of equation (37). The systematic form of the parity-check matrix displays t nonzero real row vectors, generically labeled $\underline{p}_s$, each with $(\delta+1)$ elements while the negative of the identity matrix, I_t , appears on the right part of the partitioned matrix

PARITY-CHECK MATRIX

$$H = \left[\begin{array}{c|c} \underline{p}_1 & \\ \underline{p}_2 & \\ - & \\ - & -\mathbf{I}_t \\ | & \\ \underline{p}_t & \end{array} \right] \qquad (38)$$

The partitioned vector on the left side of equation (37) is encoded by appending t parity positions contained in a parity-check vector $\underline{r}(j)$;

$$\underline{r}(j) = \begin{bmatrix} r_1(j) \\ r_2(j) \\ - \\ \cdot \\ r_t(j) \end{bmatrix} \qquad (39a)$$

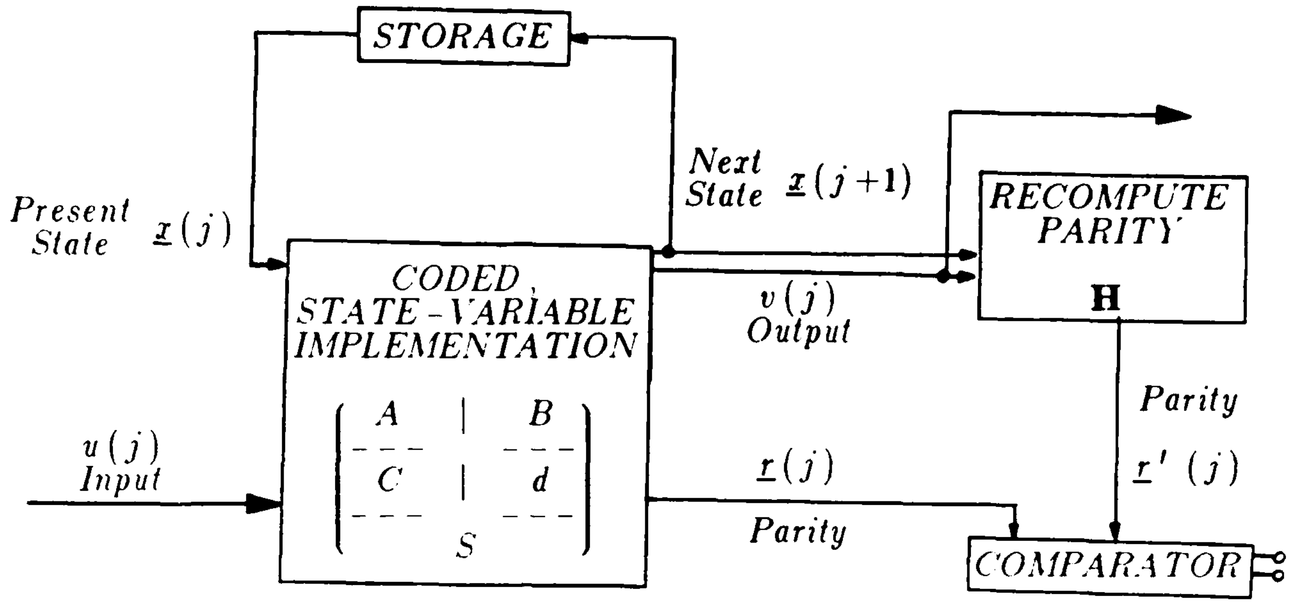
where each component comes from a vector inner product with parity-check rows from H.

$$r_i(j) = \underline{p}_i \begin{bmatrix} \underline{x}(j+1) \\ \\ v(j) \end{bmatrix} \qquad ; i=1,2,\ldots,t. \qquad (39b)$$

The theory in Corollary 1 [12] demonstrates that equivalently each column of the partitioned matrix on the right of equation (37) may be encoded producing t parity defined position at the bottom of each column. The system equation for the internally encoded digital filter contains a $[t\times(\delta+1)]$ matrix, S, representing the respective column encodings just outlined.

$$\begin{bmatrix} \underline{x}(j+1) \\ \hline v(j) \\ \hline \underline{r}(j) \end{bmatrix} = \left[\begin{array}{c|c} A & B \\ \hline C & d \\ \hline \multicolumn{2}{c}{S} \end{array} \right] \begin{bmatrix} \underline{x}(j) \\ \hline u(j) \end{bmatrix} \qquad (40)$$

A system configuration incorporating an error-detection comparator subsystem is shown in Figure 5. The elements in $\underline{r}'(j)$ are the recomputed parity samples which should correspond with similarly indexed elements generated as part of expanded matrix equation (40). It is instructive to examine the outputs related to parity samples in either $\underline{r}(j)$ or $\underline{r}'(j)$. Without lose of generality, assume that


STORAGE
Present State $\underline{x}(j)$
Next State $\underline{x}(j+1)$
RECOMPUTE PARITY
H
CODED, STATE-VARIABLE IMPLEMENTATION
$\begin{pmatrix} A & | & B \\ C & | & d \\ & S & \end{pmatrix}$
$v(j)$ Output
$u(j)$ Input
$\underline{r}(j)$
Parity
Parity
$\underline{r}'(j)$
COMPARATOR

BLOCK CODED, STATE-VARIABLE APPROACH TO PROTECTED FILTER

Figure 5

the beginning initial state of the filter is zero. The Z transforms of quantities related to the parity samples associated with lower part of equation (40) may be written using upper case letters to signify Z transforms of corresponding sample sequences in (40).

$$\underline{R}(Z) = S \begin{bmatrix} X(Z) \\ U(Z) \end{bmatrix} \tag{41}$$

However, the next-state items are expressible through the usual system transfer matrix which includes the poles and zero of the original filter.

$$X(A) = (ZI - A)^{-1} B\ U(Z) \tag{42}$$

The desired relationship governing the overall system response for the parity samples follows by combining equations (41) and (42).

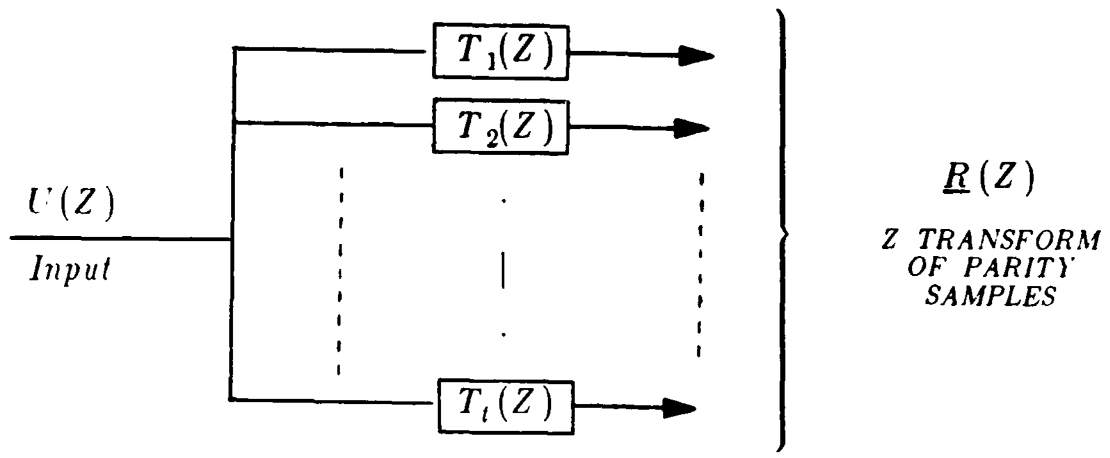
$$\underline{R}(Z) = S \begin{bmatrix} (ZI - A)^{-1} B \\ 1 \end{bmatrix} U(Z) \tag{43}$$

The right side defines t separate parity filters, one for each component $T_j(Z)$ of $\underline{R}(Z)$.

$$\underline{R}(Z) = \begin{bmatrix} T_1(Z) \\ T_2(Z) \\ - \\ \cdot \\ | \\ \cdot \\ T_t(Z) \end{bmatrix} U(Z) = S \begin{bmatrix} (ZI - A)^{-1} B \\ 1 \end{bmatrix} U(Z) \tag{44}$$

The parallel filter viewpoint of this parity protection scheme is shown in Figure 6.

Several aspects of this protection technique are noteworthy. Parity samples are calculated at each sample instant permitting no efficient decimation. The filter poles are present in each parity calculation channel. Furthermore the error performance of the code rests on the Van der Monde nature of the parity-check matrix [12], producing codes with construction and distance properties analogous to BCH codes [3,4]. In general there are no extra parameters to adjust for canceling poles in the parity channels' system transfer functions. The convolutional code appears more flexible and efficient.



EQUIVALENT PARITY SAMPLE CALCULATIONS IN STATE ENCODED FILTER SYSTEM

Figure 6

REFERENCES

[1] John Wakerly, *Error Detecting Codes, Self-Checking Circuits and Applications.* New York: North-Holland, 1978.

[2] D. K. Pradhan, Editor, *Fault-Tolerant Computing Theory and Techniques, Volume I.* Englewood Cliffs: Prentice-Hall, 1986.

[3] S. Lin and D. J. Costello, Jr., *Error Control Coding Fundamentals and Applications.* Englewood Cliffs: Prentice-Hall, 1983.

[4] R. E. Blahut, *Theory and Practice of Error Control Codes* . Reading, Massachusetts: Addison-Wesley, 1983.

[5] T. G. Marshall, Jr., " Coding of Real-Number Sequences for Error Correction: A Digital Signal Processing Problem ", *IEEE Journal on Selected Areas In Communications* , vol. SAC-2, pp. 381-392, 1984.

[6] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing* . Englewood Cliffs: Prentice-Hall, 1983.

[7] N. K. Bose, *Digital Filters Theory and Applications* . New York: North-Holland, 1985.

[8] S. A. Tretter, *Introduction to Discrete-Time Signal Processing.* New York: Wiley, 1976.

[9] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing.* Englewood Cliffs: Prentice-Hall, 1975.

[10] The Mathlab Group, " MACSYMA Reference Manual ", MIT Laboratory for Computer Science, Version Ten, 1983.

[11] J. P. Dugre, A. A. L. Beex, and L. L. Scharf, " Generating Covariance Sequences and the Calculation of Quantization and Roundoff Error Variances in Digital Filters ", *IEEE Transactions on Acoustics, Speech and Signal Processing* , vol. ASSP-28, pp. 102-104, 1980.

[12] J. Y. Jou and J. A. Abraham, " Fault-Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent Computing Structures ", *Proceedings of IEEE*, vol. 74, pp. 732-741, 1984.

[13] I. N. Herstein, *Topics In Algebra* . New York: Blaisdell, 1964.